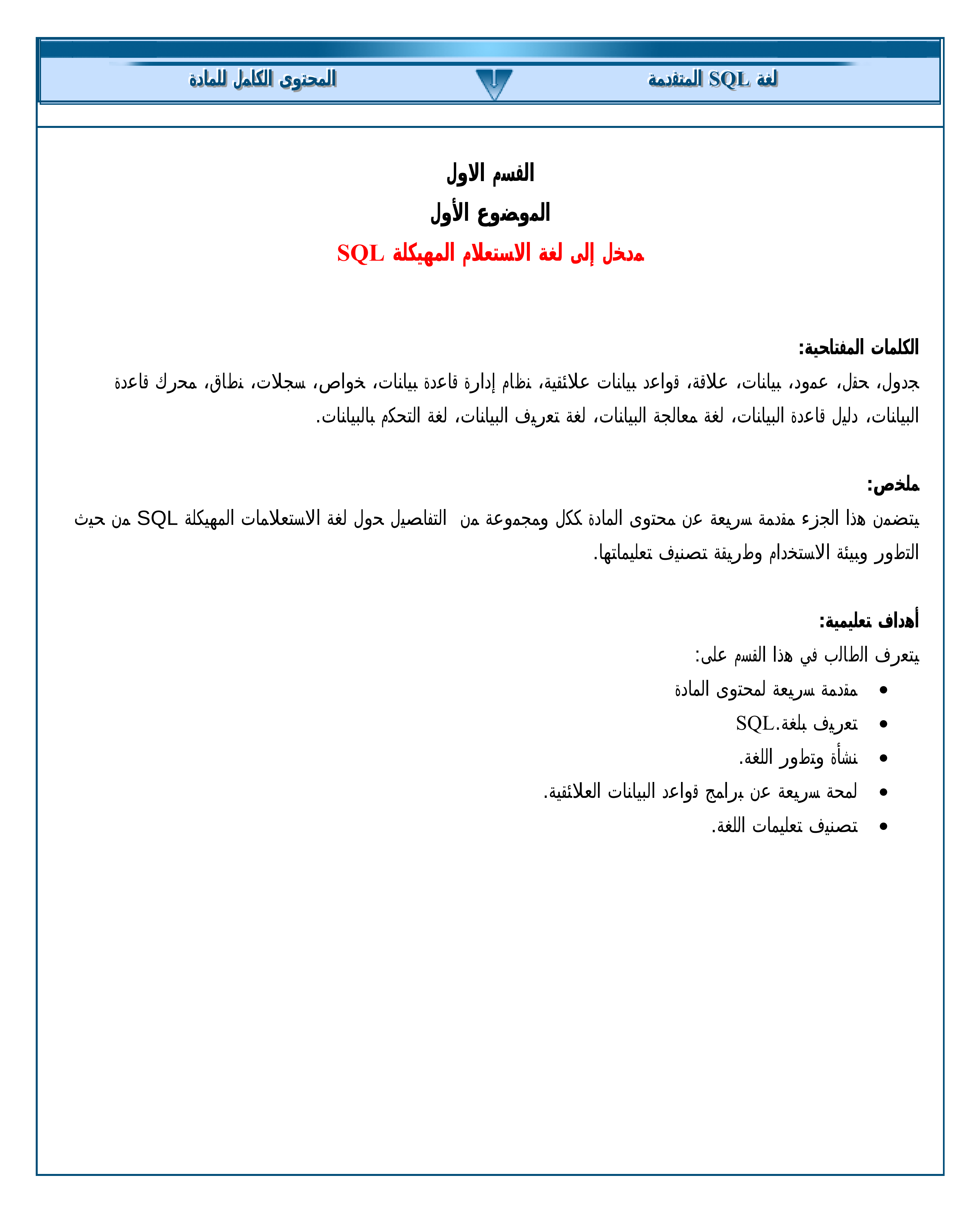


القسم الاول

الموضوع الأول

مدخل إلى لغة الاستعلام المهيكلة SQL

الكلمات المفتاحية:

جدول، حقل، عمود، بيانات، علاقة، قواعد بيانات علائقية، نظام إدارة قاعدة بيانات، خواص، سجلات، نطاق، محرك قاعدة البيانات، دليل قاعدة البيانات، لغة معالجة البيانات، لغة تعريف البيانات، لغة التحكم بالبيانات.

ملخص:

يتضمن هذا الجزء مقدمة سريعة عن محتوى المادة ككل ومجموعة من التفاصيل حول لغة الاستعلامات المهيكلة SQL من حيث التطور وبيئة الاستخدام وطريقة تصنيف تعليماتها.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- مقدمة سريعة لمحتوى المادة
- تعريف بلغة SQL.
- نشأة وتطور اللغة.
- لمحة سريعة عن برامج قواعد البيانات العلائقية.
- تصنيف تعليمات اللغة.

مدخل إلى محتوى المادة

تُعرف لغة الاستعلام المهيكلة SQL بأنها لغة التعامل مع قواعد البيانات وتعتمد عليها كافة التطبيقات التي تتعامل مع قواعد البيانات العلائقية.

تهدف هذه المادة إلى تغطية أوجه استخدام لغة SQL وتوضيح إمكانياتها التي تساعد على استثمار أفضل وأسهل لقواعد البيانات.

سنبدأ بمقدمة عن اللغة ثم نستعرض تعليمات اللغة ابتداءاً من أبسط التعليمات، خصوصاً تلك التي تغطي عمليات الإضافة والحذف والاستعلام عن السجلات، مروراً بالتوابع وعمليات ربط الجداول والاستعلامات الفرعية، وانتهاءاً بإدارة الجداول وإدارة المستخدمين والسماحيات.

سنتعرف أيضاً على بعض أجزاء اللغة الإجرائية PL/SQL وعلى كيفية إدارة الأخطاء وتجاوزها.

سيتم دعم كافة الأفكار بأمثلة عملية وواضحة إضافةً إلى مشروع يساعد على تجميع كافة الأفكار ووضعها ضمن قالب تطبيقي مباشر.

ما هي لغة SQL

تُعرف SQL (تلفظ SEQUEL) بأنها لغة تساعد على التعامل مع قواعد البيانات العلائقية. وقد تم تطوير لغة SQL انطلاقاً من النموذج العلائقي لـ كودد(CODD) المعتمد على الجبر العلائقي.

تُعتبر لغة SQL لغة معيارية تتبع معايير ISO وANSI، وقد تم دعمها من قبل عدد كبير من الشركات العاملة في مجال تطوير برامج إدارة قواعد البيانات العلائقية مثل Microsoft وOracle



مراحل تطور لغة SQL

مرت اللغة بمراحل التطور التالية منذ نشأتها عام 1970:

السنة	المرحلة
1970	نموذج كود المعتمد على الجبر العلائقي
1974	استخدام sql مع أجهزة mainFrame
1979	دعم SQL من قبل النسخة الأولى التجارية من برنامج إدارة قواعد البيانات oracle
1987	أطلقت منظمات ISO و ANSI للمعايير أول معيار لـ SQL هو SQL89
1991	تم نشر المعيار 92- SQL
1999	تم نشر المعيار 99- SQL أو ما يطلق عليه SQL3

المعيار الذي سنقوم باستخدامه في مادتنا هو المعيار SQL 99.

انتبه:

معيارية ولكن!!!:

بالرغم من كون SQL لغة معيارية فقد تم اعتمادها من قبل شركات تطوير أنظمة إدارة قواعد البيانات بأشكال مختلفة وتم إقحام بعض الإضافات أو تعديل بعض التعليمات. على كل حال، بقيت اللغة في غالبها تدعم التعليمات الرئيسية.

سننوه إلى بعض هذه الاختلافات في بعض الأمثلة التطبيقية التي نقدمها.

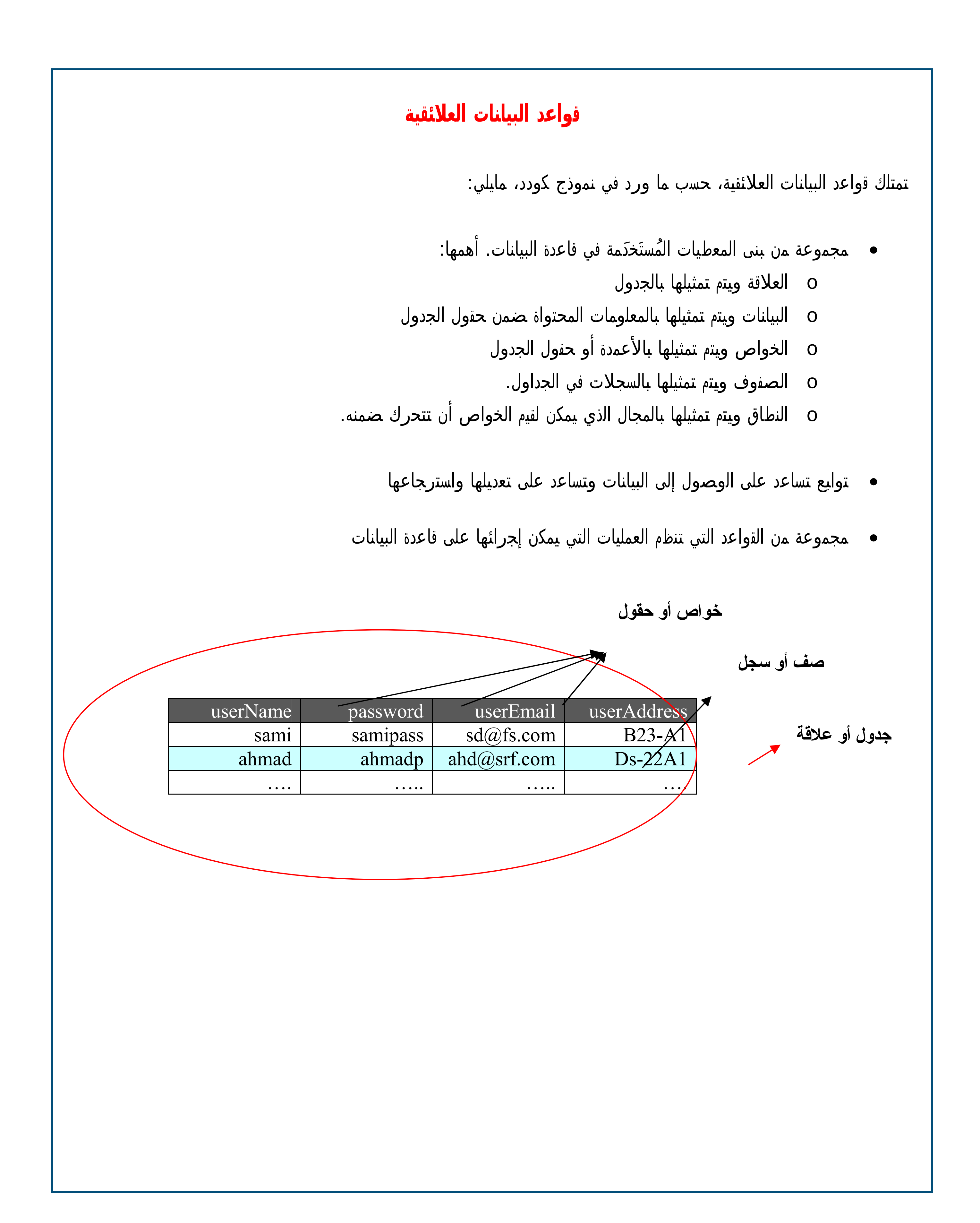
قواعد البيانات العلائقية

تمتلك قواعد البيانات العلائقية، حسب ما ورد في نموذج كودد، مايلي:

- مجموعة من بنى المعطيات المُستَخدَمة في قاعدة البيانات. أهمها:
  - o العلاقة ويتم تمثيلها بالجدول
  - o البيانات ويتم تمثيلها بالمعلومات المحتواة ضمن حقول الجدول
  - o الخواص ويتم تمثيلها بالأعمدة أو حقول الجدول
  - o الصفوف ويتم تمثيلها بالسجلات في الجداول.
  - o النطاق ويتم تمثيلها بالمجال الذي يمكن لقيم الخواص أن تتحرك ضمنه.

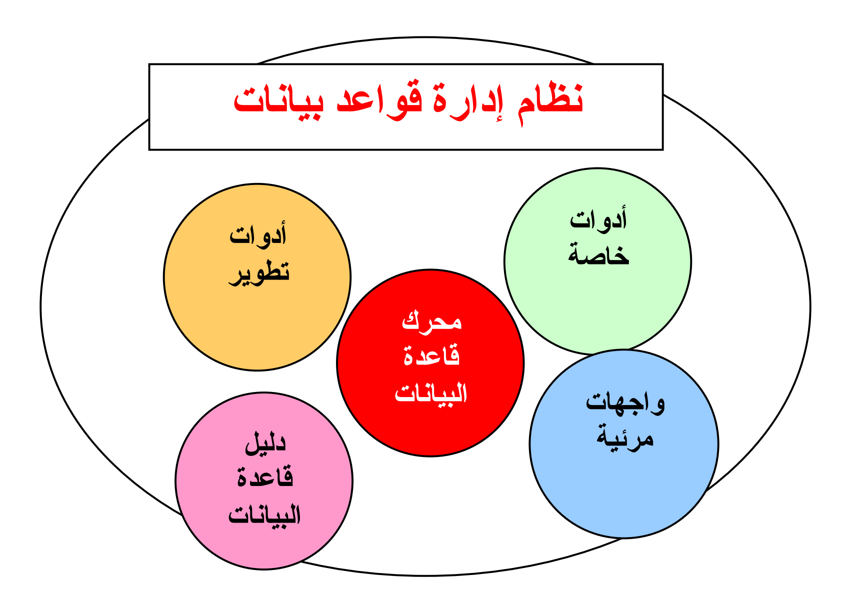
- توابع تساعد على الوصول إلى البيانات وتساعد على تعديلها واسترجاعها

- مجموعة من القواعد التي تنظم العمليات التي يمكن إجرائها على قاعدة البيانات



userName	password	userEmail	userAddress
sami	samipass	sd@fs.com	B23-A1
ahmad	ahmadp	ahd@srf.com	Ds-22A1
….	…..	…..	….

قواعد البيانات العلائقية أم أنظمة إدارة قواعد البيانات العلائقية!!



من الأخطاء الشائعة، إطلاق اسم قواعد البيانات على أنظمة إدارة قواعد البيانات العلائقية. فأنظمة SQL server، وOracle، وMS-Access هي أنظمة إدارة قواعد بيانات وليست قواعد بيانات.

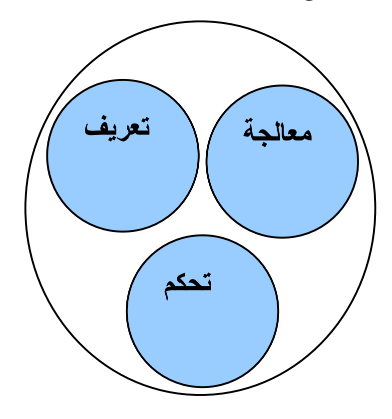
تتكون هذه الأنظمة ممايلي:

- محرك قاعدة البيانات ويعتبر العنصر الأهم المسؤول عن تخزين البيانات ومعالجتها
- دليل قاعدة البيانات الذي يحتوي كافة المعلومات التي تخص قاعدة البيانات والجداول والحقول وأنواعها
- واجهات مرئية لإدارة المعطيات وتقديم نماذج وتقارير واستعلامات
- أدوات خاصة بقواعد البيانات تشمل التوليد التلقائي لمخططات قواعد البيانات
- أدوات تطوير التطبيقات

SQL

تعمل SQL بمبدأ توجيه طلب إلى محرك قاعدة البيانات والحصول على جواب منه بحيث نحصل على مجموعة نتائج.
توفر SQL مجموعة من التعليمات بحيث يمكن تقسيمها إلى ثلاث لغات فرعية:

- لغة معالجة البيانات: Select, insert, delete, update
- لغة تعريف البيانات: create table, drop table, alter table, create index
- لغة التحكم بالبيانات: grant, revoke



تعمل SQL بمبدأ توجيه طلب إلى محرك قاعدة البيانات والحصول على جواب من محرك قاعدة البيانات الذي يُرجع مجموعة نتائج.
توفر SQL مجموعة من التعليمات بحيث يمكن تقسيمها إلى ثلاث لغات فرعية:

لغة معالجة البيانات التي تتضمن التعليمات الخاصة باستعادة البيانات وتعديلها مثل:
Select: وهي مخصصة لقراءة البيانات واستخلاصها من قاعدة البيانات.
Insert: وهي مخصصة لإضافة سجلات جديدة إلى قاعدة البيانات.
Delete: وهي مخصصة لحذف سجل أو مجموعة سجلات من قاعدة البيانات.
Update: وهي مخصصة لتعديل سجل أو مجموعة من السجلات في قاعدة البيانات.

لغة تعريف البيانات المخصصة لتعريف بنية البيانات، وتتضمن تعليمات مثل:
Create table: وهي مسؤولة عن توليد جدول
drop table: وهي مسؤولة عن حذف جدول
alter table: وهي مسؤولة عن تعديل جدول
create index: وهي مسؤولة عن توليد مؤشرات

لغة التحكم بالبيانات التي تُستخدم للتحكم وضبط السماحيات على قاعدة البيانات مثل:
grant
revoke

الموضوع الثاني

لغة معالجة المعطيات

الكلمات المفتاحية:

تعليمة، صيغة، جدول، حقل، سجل، عمود، استعلام، استعلام فرعي.

ملخص:

تم تصنيف تعليمات لغة SQL إلى ثلاثة أصناف رئيسية ضمن ثلاث لغات فرعية: لغة معالجة المعطيات، ولغة تعريف المعطيات، ولغة التحكم بالمعطيات. ومن بين هذه التعليمات تعتبر تعليمات معالجة المعطيات الأكثر استخداماً. سنستعرض تعليمات المعالجة والتي تشمل SELECT,INSERT,DELETE,UPDATE وبعض الكلمات المفتاحية المُستخدمة معها.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- استخدام تعليمة SELECT لاستعادة البيانات من قاعدة البيانات.
- استخدام تعليمة INSERT لإدراج سجلات ضمن جدول في قاعدة البيانات.
- استخدام تعليمة DELETE لحذف سجل أو مجموعة من السجلات.
- استخدام تعليمة UPDATE لتعديل سجل أو مجموعة من السجلات.
- التعرف على الإطارالتطبيقي لاستخدام هذه التعليمات مع الكلمات المفتاحية المرافقة لها.

تعليمة Select 1

تُعتبر تعليمة Select من أشهر تعليمات اللغة وأكثرها استخداماً. تُستخدَم هذه التعليمة لاستعادة وانتقاء مجموعة من البيانات من قاعدة البيانات وذلك بإعادة جدول يحتوي مجموعة البيانات المطلوبة

تأخذ تعليمة Select الصيغة:

```
Select [ field1,field2,…] from [table_name];
```

- تُستخدم إشارة * كبديل لأسماء الحقول (عادة لاننصح باستخدامها في الحالات التطبيقية لأنها تُحمل برنامج إدارة قاعدة البيانات عبء تحديد الحقول وتحديد عددها وأسماءها).
- يُستخدم تعبير Distinct لاستعادة جميع السجلات مع إلغاء التكرار في السجلات المعادة.
- يُستخدم التعبير Order by لترتيب السجلات المعادة ترتيباً تصاعدياً أو تنازلياً حسب التعبير المرافق المستخدم: ASC للترتيب التصاعدي أو DESC للترتيب التنازلي.
- في حال الرغبة باستخدام أسماء بديلة لحقول جدول القيم المعادة نستخدم التعبير AS.

مثال:

إذا كان لدينا قاعدة بيانات تحتوي جدول يدعى Users وأردنا استعادة بيانات حقلي username و password من جميع سجلات هذا الجدول، تكون عبارة SQL كما يلي:

```
Select username, password from Users
```

لاستعادة جميع السجلات من الجدول:

```
Select * from Users
```

لاستعادة جميع بيانات الحقل UserName مع إلغاء التكرار في السجلات المُعادة:

```
Select Distinct UserName from Users
```

لاستعادة مجموعة من السجلات مرتبة ترتيباً تصاعدياً وفق أحد الحقول:

```
Select userName, Password from users order by userName  ASC
```

لاستخدام اسم بديل للحقل userName في جدول القيم المعادة هو Names نكتب التعبير:

```
Select username As Names from users
```

تُعتبر تعليمة Select من أشهر تعليمات اللغة وأكثرها استخداماً. تُستخدَم هذه التعليمة لاستعادة وانتقاء مجموعة من البيانات من قاعدة البيانات وذلك بإعادة جدول يحتوي مجموعة البيانات المطلوبة

- تُستخدم إشارة * (star) كبديل لأسماء الحقول (عادة لاننصح باستخدامها في الحالات التطبيقية لأنها تُحمل برنامج إدارة قاعدة البيانات عبء تحديد الحقول وتحديد عددها وأسماءها).
- يُستخدم تعبير Distinct لاستعادة جميع السجلات مع إلغاء التكرار في السجلات المعادة.
- يُستخدم التعبير Order by لترتيب السجلات المعادة ترتيباً تصاعدياً أو تنازلياً حسب التعبير المرافق المستخدم: ASC للترتيب التصاعدي أو DESC للترتيب التنازلي.
- في حال الرغبة باستخدام أسماء بديلة لحقول جدول القيم المُعادة نستخدم التعبير AS.

تعليمة SELECT 2

الكلمة المفتاحية WHERE:

نستخدم الكلمة المفتاحية Where مع تعليمة Select لاستعادة مجموعة من السجلات التي تحقق شرط أو مجموعة من الشروط التي نعبر عنها بعبارة شرطية.

تستخدم الكلمة المفتاحية where الصيغة:

```
Select * from users where condition;
```

- تُعيد العبارة الشرطية قيمة منطقية (صح أو خطأ)

- يمكن للعبارة الشرطية أن تتضمن عمليات مقارنة مثل (=< , =< , > , < , < > , =) ويتم ضم السجل الذي يحققها إلى جدول النتائج.

الكلمات المفتاحية Like betweeng

- تُستخدم الكلمة المفتاحية Like ضمن العبارة الشرطية، كشرط لوجود مثيل. غالباً ما تُستخدَم هذه الكلمة مع إشارة (%)، التي تضاف إلى القيمة التي نبحث عن مثيلاتها، كبديل عن أي رقم من الأرقام أوالأحرف.

- تُستخدم الكلمة المفتاحية Between ضمن العبارة الشرطية، كشرط لوجود قيمة محصورة بين قيمتين محددتين

- تقبل الكلمة المفتاحية where أكثر من شرط يفصل بينها عمليات منطقية مثل AND أو OR ويمكن أن يسبق الشرط العملية NOT لنفيه.

مثال:

إذا أردنا الحصول على قائمة جميع السجلات التي تحتوي على السلسلة ‘am’ (بمطابقة جزئية أو بمطابقة كاملة) في حقل اسم المستخدم يمكن استخدام تعليمة Select التالية:

```
Select * from users where userName like ‘%am%’;
```

للحصول على قائمة جميع السجلات التي تنحصر فيها قيمة حقل العمر بين 15 و25 يمكن كتابة تعليمة Select التالية:

```
Select * from users where userAge between 15 and 25;
```

إذا أردنا الحصول على قائمة جميع السجلات التي تحتوي على السلسلة ‘am’ (بمطابقة جزئية أو بمطابقة كاملة) في حقل اسم المستخدم، والتي تنحصر فيها قيمة حقل العمر بين 15 و25 يصبح المثال كما يلي:

```
Select * from users where userName like ‘%am%’
 And
userAge between 15 and 25;
```

الكلمة المفتاحية WHERE:

نستخدم الكلمة المفتاحية Where مع تعليمة Select لاستعادة مجموعة من السجلات التي تحقق شرط أو مجموعة من الشروط التي نعبر عنها بعبارة شرطية.

- تُعيد العبارة الشرطية قيمة منطقية (صح أو خطأ)

- يمكن للعبارة الشرطية أن تتضمن عمليات مقارنة مثل (=> , =< , < , > , <> , =) ويتم ضم السجل الذي يحققها إلى جدول النتائج.

الكلمات المفتاحية Like وbetween

- تُستخدم الكلمة المفتاحية Like ضمن العبارة الشرطية، كشرط لوجود مثيل. غالباً ما تُستخدَم هذه الكلمة مع إشارة (%)، التي تضاف إلى القيمة التي نبحث عن مثيلاتها، كبديل عن أي رقم من الأرقام أوالأحرف.
- تُستخدم الكلمة المفتاحية Between ضمن العبارة الشرطية، كشرط لوجود قيمة محصورة بين قيمتين محددتين
- تقبل الكلمة المفتاحية where أكثر من شرط يفصل بينها عمليات منطقية مثل AND أو OR ويمكن أن يسبق الشرط العملية NOT لنفيه.

تعليمة DELETE

تقوم تعليمة Delete بحذف سجل أو مجموعة من السجلات من جدول ما.

تأخذ تعليمة Delete الصيغة:

```
Delete from [table_name]
```

مثال:

للحذف من جدول users نستخدم تعليمة Delete التالية:

```
Delete from Users
```

تُعتبر التعليمة الواردة في المثال السابق خطرة لأنها ستقوم بحذف جميع السجلات من الجدول Users. لذا نحتاج إلى الكلمة المفتاحية Where لتحديد شرط حذف هذه السجلات.

فإذا كنا نريد حذف السجل الخاص بالمستخدم ذي اسم الدخول 'Ahmed' يصبح مثالنا كالتالي:

```
Delete from Users where username='Ahmed';
```

التعليمة Delete

تقوم تعليمة Delete بحذف سجل أو مجموعة من السجلات من جدول ما.

تعليمة INSERT

تُستخدم تعليمة Insert لإدراج سجل في جدول محدد.

تأخذ تعليمة Insert الصيغة:

```
insert into table_name values ( value1,value2,value3,…);
```

حيث تُمثل value1,..value n القيم التي سوف يتم تخزينها في السجل تبعاً لترتيب حقوله.

كما يمكننا استخدام صيغة بديلة تُمكننا من تحديد حقول السجل التي نود إدراج البيانات فيها فقط:

```
Insert into table_name (field1,field2….)
 values (value1,value2,..);
```

يمكن لتعليمة Insert إدراج أكثر من سجل بأمر واحد ولكن ستحتاج إلى استخدام ما ندعوه الاستعلامات الفرعية (Sub queries) التي سنأتي على ذكرها لاحقاً.

مثال:

لإدراج سجل كامل في الجدول users نستخدم الصيغة:

```
insert into Users values
('adel','adelPassword',33,'adel@yahoo.com');
```

أما الحل البديل الذي يُمكننا من تحديد حقول السجل التي نود إدراج البيانات ضمنها وترتيبها فهو:

```
insert into Users (username,password)
values ('adel','adelPassword');
```

لإدراج جميع النتائج، التي أعادها الاستعلام عن جميع حقول الجدول otherUserTable، ضمن الجدول users نستخدم العبارة:

```
Insert into users select * from OtherUserTable
```

تُستخدم تعليمة Insert لإدراج سجل في جدول محدد

يمكن لتعليمة Insert إدراج أكثر من سجل بأمر واحد ولكن ستحتاج إلى استخدام ما ندعوه الاستعلامات الفرعية (Sub queries) التي سنأتي على ذكرها لاحقاً.

تعليمة Update

تُستخدم تعليمة Update من تعديل البيانات في سجل أو في مجموعة من السجلات.

تأخذ تعليمة Update الصيغة:

```
Update table_name Set
Field1= new_field_value1 ,
Field2= new_field_value 2;
```

يمكن استخدام الكلمة المفتاحية where مع تعليمة Update لتصبح الصيغة:

```
Update table_name Set
Field1= new_field_value1 ,
Field2= new_field_value 2
Where condition ;
```

مثال:

```
Update Users set username='sami' , password='sami pass'
```

سيقوم مثالنا بتعديل قيمة اسم المستخدم وكلمة العبور في كل السجلات وفقاً للقيم المعطاة، لذلك لابد هنا من الانتباه إلى استخدام التعبير where ليعطي شرط التعديل كما يلي:

```
Update Users set password='sami pass' where username='sami'
```

تُستخدم تعليمة Update من تعديل البيانات في سجل أو في مجموعة من السجلات. ويمكن استخدام الكلمة المفتاحية where مع تعليمة Update لتحديد شروط التعديل.

بعض الملاحظات العملية

- تتطلب بعض برامج إدارة قواعد البيانات إنهاء كل تعليمة SQL بـ (;) بينما يضيفها البعض الآخر تلقائياً.
- من المهم استخدام تعليقات SQL وخصوصاً عند كتابة نصوص SQL تحتوي على عدد كبير من الأسطر والتعليمات. يمكن إضافة التعليق كما يلي:

```
Select * from users;   -- this is the comment
```

- من المهم تلافي استخدام أسماء أعمدة (حقول) حاوية على فراغات. أما في الحالات الاضطرارية، فيمكن استخدام إشارات تنصيص أو أقواس مربعة لإحاطة اسم الحقل (أقواس في oracle وإشارة تنصيص في Acess وSQL server) مثال:

```
Select [user name] from users ;
```

تتطلب بعض برامج إدارة قواعد البيانات إنهاء كل تعليمة SQL بـ (;) بينما يضيفها البعض الآخر تلقائياً.
ومن المهم استخدام تعليقات SQL وخصوصاً عند كتابة نصوص SQL تحتوي على عدد كبير من الأسطر والتعليمات.
كذلك، من المهم تلافي استخدام أسماء أعمدة (حقول) حاوية على فراغات. أما في الحالات الاضطرارية، فيمكن استخدام إشارات تنصيص أو أقواس مربعة لإحاطة اسم الحقل (أقواس في oracle وإشارة تنصيص في Acess وSQL server).

القسم الثاني

التوابع في SQL

الكلمات المفتاحية:

تابع، تابع تجميعي، تابع درجي، قيمة، بيانات، حقل، جدول.

ملخص:

التوابع في SQL عديدة تم تصنيفها بحسب عدد القيم المُدخلة إلى نوعين: التوابع التجميعية والتوابع الدرجية. سيغطي هذا الجزء التوابع التجميعية والكلمات المفتاحية والتعابير المُستخدمة معها.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- مفهوم التوابع التجميعية والمدرجة
- الصيغة والتطبيق للتوابع التجميعية المختلفة
- التعابير المستخدمة مع التوابع التجميعية وكيفية استخدامها

التوابع في SQL

التابع هو عبارة عن تعبير رياضي يأخذ مجموعة من قيم الدخل التي ندعوها مُعاملات، ويعيد قيمة خرج وحيدة ندعوها قيمة التابع. تتعلق قيمة التابع (أي الخرج) بمعاملاته (أي بالدخل)، كحال التابع الذي يقوم بحساب مجموع قيم عددية.

التوابع في SQL

تُقسَم توابع SQL، حسب معيار SQL-99، إلى نوعين:

- التوابع التجميعية: مثال: $f(x,y,z)=x+y+z$
- التوابع الدرجية: مثال: $f(x) = |x|$

التوابع في SQL
(حسب معيار SQL-99)

التوابع التجميعية
التوابع الدرجية

التابع هو عبارة عن تعبير رياضي يأخذ مجموعة من قيم الدخل التي ندعوها مُعاملات، ويعيد قيمة خرج وحيدة ندعوها قيمة التابع. تتعلق قيمة التابع (أي الخرج) بمُعاملاته (أي بالدخل)، كحال التابع الذي يقوم بحساب مجموع قيم عددية.

التوابع في SQL

تُقسَم توابع SQL حسب معيار SQL-99 إلى نوعين:

- التوابع التجميعية: وهي التوابع التي تأخذ كمُعاملات مجموعة من القيم وتعيد قيمة وحيدة، مثل التابع الذي يحسب مجموع أعداد حقيقية.

- التوابع الدرجية وهي التوابع التي تأخذ مُعاملاً وحيداً وتُعيد قيمة وحيدة، مثل تابع القيمة المُطلقة لعدد حقيقي.

توابع SQL التجميعية

من أهم توابع SQL التجميعية التوابع التالية:

التابع	استخدامه
AVG(expression)	يقوم بحساب معدل القيم لحقل معين
COUNT(expression)	يقوم بحساب عدد البيانات الخاصة بحقل معين
MIN(expression)	يقوم بإعادة القيمة الصغرى من قيم حقل معين
MAX(expression)	يقوم بإعادة القيمة العظمى من قيم حقل معين
SUM(expression)	يقوم بحساب مجموع قيم حقل معين

من أهم توابع SQL التجميعية التوابع التالية:

التابع *AVERAGE* الذي يُستخدم لحساب المتوسط الحسابي لقيم حقل معين.

التابع *COUNT* الذي يُستخدم لحساب عدد البيانات الخاصة بحقل مُعين.

التابع *MIN* الذي يُستخدم لحساب القيمة الصغرى من قيم حقل معين.

التابع *MAX* الذي يُستخدم لحساب القيمة العظمى من قيم حقل معين.

التابع *SUM* الذي يإستخدم لحساب مجموع قيم حقل معين.

سنستعرض صيغة كل تابع من التوابع السابقة وأسلوب عمله بالتفصيل في الفقرات اللاحقة.

التابع AVG

يحسب تابع **AVG** المتوسط الحسابي لقيم حقل معين وذلك بتقسيم مجموع قيم هذا الحقل على عددها.

صيغة التابع:

```
select avg([ALL | DISTINCT]column_name) from table_name
```

- يُستخدم الخيار All للحصول على المتوسط الحسابي لجميع القيم بما فيها القيم المكررة. يُعتبر هذا الخيار هو الخيار التلقائي في حال عدم تحديد أي من الخيارين Distinct أو All.

- يُستخدم الخيار Distinct للحصول على المتوسط الحسابي لجميع القيم، مع استبعاد أي تكرار لقيمة ما.

مثال:

إذا كان لدينا الجدول علامات الطلاب grades الذي يحتوي الحقول studentName وstudentGrade وstudentClass.

للحصول على المتوسط الحسابي لجميع الطلاب، نستخدم التعبير:

```
select avg(studentGrade) from grades
```

إذا أردنا الحصول على المتوسط الحسابي لعلامات الطالب "عادل" مع استبعاد أي تكرار لعلامة من علاماته، يصبح التعبير:

```
select avg(distinct studentGrade) form grades where studentName =
'adel'
```

انتبه:

- قد لا تعمل خيارات All وDistinct على قواعد بيانات MS Access.
- قد تختلف القيمة التي يعيدها تابع AVG من نظام إدارة قواعد بيانات إلى آخر بسبب تحويل النتيجة في الغالب إلى نفس نوع الحقل الذي تم تطبيق التابع عليه.

يحسب تابع **AVG** المتوسط الحسابي لقيم حقل معين وذلك بتقسيم مجموع قيم هذا الحقل على عددها.

- يُستخدم الخيار All للحصول على المتوسط الحسابي لجميع القيم بما فيها القيم المكررة. يُعتبر هذا الخيار هو الخيار التلقائي في حال عدم تحديد أي من الخيارين Distinct أو All.

- يُستخدم الخيار Distinct للحصول على المتوسط الحسابي لجميع القيم، مع استبعاد أي تكرار لقيمة ما.

التابع COUNT

يحسب التابع COUNT عدد البيانات الموجودة في الجدول من أجل حقل معين.

صيغة التابع:

```
select count([* | ALL | DISTINCT]column_name) from table_name
```

- يُستخدم الخيار All عندما نريد الحصول على عدد البيانات الموجودة في الجدول، بالنسبة لحقل معين، مع استبعاد القيم التي تساوي Null. يُعتبر هذا الخيار هو الخيار التلقائي في حال عدم تحديد أي من الخيارين Distinct أو All.

- يُستخدم الخيار Distinct عندما نريد الحصول على عدد البيانات الموجودة في الجدول، بالنسبة لحقل معين، مع استبعاد القيم التي تساوي Null واستبعاد أي تكرار في قيمة ما.

- يُستخدم الخيار * عندما نريد الحصول على عدد البيانات الموجودة في الجدول بالنسبة لحقل معين بما فيها البيانات ذات القيمة Null.

مثال:

إذا كان لدينا الجدول علامات الطلاب grades الذي يحتوي الحقول studentName وstudentGrade وstudentClass. وإذا أردنا الحصول على عدد الطلاب نستخدم التعليمة:

```
select count(*) from grades
```

لكن المثال السابق سيحسب عدد الطلاب اعتماداً على السجلات التي قد تحتوي على القيم Null. لذلك إذا أردنا الحصول على عدد الطلاب توجب علينا حساب عدد البيانات الموجودة في الجدول بالنسبة لحقل اسم الطالب شرط ألا يكون اسم الطالب مساوياً لـ Null (مما يعني أن الطالب موجود فعلاً)، تصبح التعليمة عندها:

```
select count(all studentName) from grades
```

لكن، قد يكون أحد أسماء الطلاب مكرراً، لذا نستخدم التعليمة التالية لنحصل على عدد الطلاب الحقيقي:

```
select count(distinct studentName) from grades
```

انتبه:

قد لا تعمل خيارات All وDistinct على قواعد بيانات MS Access.

يحسب التابع COUNT عدد البيانات الموجودة في الجدول من أجل حقل معين.

- يُستخدم الخيار All عندما نريد الحصول على عدد البيانات الموجودة في الجدول، بالنسبة لحقل معين، مع استبعاد القيم التي تساوي Null. يُعتبر هذا الخيار هو الخيار التلقائي في حال عدم تحديد أي من الخيارين Distinct أو All.

- يُستخدم الخيار Distinct عندما نريد الحصول على عدد البيانات الموجودة في الجدول، بالنسبة لحقل معين، مع استبعاد القيم التي تساوي Null واستبعاد القيم المكررة.

- يُستخدم الخيار * عندما نريد الحصول على عدد البيانات الموجودة في الجدول، بالنسبة لحقل معين، بما فيها البيانات ذات القيمة Null.

التابع MIN والتابع MAX

يُعيد التابع MIN القيمة الصغرى من قيم حقل معين.

صيغة التابع:

```
select min(column_name) from table_name
```

يُعيد التابع MAX القيمة العظمى من قيم حقل معين.

صيغة التابع:

```
select max(column_name) from table_name
```

لا تأثير للخيارين All وDistinct على التوابع MIN وMAX رغم أنه بالإمكان استخدامهما. فالقيمة العظمى أو القيمة الصغرى، لقيم حقل، تبقى نفسها حتى ولو كان هناك تكرار في قيم الحقل وحتى ولو كان هناك قيم غير محددة (أي تساوي Null).

مثال:

إذا كان لدينا الجدول علامات الطلاب grades الذي يحتوي الحقول studentName وstudentGrade وstudentClass، وإذا أردنا الحصول على أخفض علامة نستخدم التعليمة:

```
select min(studentGrade) from grades
```

و إذا أردنا الحصول على أعلى علامة نستخدم التعليمة:

```
select max(studentGrade) from grades
```

يُعيد التابع MIN القيمة الصغرى من قيم حقل معين في حين يُعيد التابع MAX القيمة العظمى من قيم حقل معين.

تجدر الإشارة إلى عدم وجود أي تأثير للخيارين All وDistinct على التوابع MIN وMAX رغم أنه بالإمكان استخدامهما. فالقيمة العظمى أو القيمة الصغرى لقيم حقل، تبقى نفسها، حتى ولو كان هناك تكرار في قيم الحقل وحتى ولو كان هناك قيم غير محددة (أي تساوي Null).

التابع SUM

يحسب التابع SUM مجموع قيم حقل معين.

صيغة التابع:

```
select sum([ALL | Distinct]column_name) from table_name
```

- يُستخدم الخيار All عندما نريد الحصول على مجموع قيم حقل معين بما فيها القيم المكررة. يُعتبر هذا الخيار هو الخيار التلقائي في حال عدم تحديد أي من الخيارين Distinct أو All.
- يستخدم الخيار Distinct عندما نريد الحصول على مجموع قيم حقل معين مع استبعاد أي تكرار في القيم.

مثال:

إذا كان لدينا الجدول علامات الطلاب grades الذي يحتوي الحقول studentName وstudentGrade وstudentClass، وإذا أردنا الحصول على مجموع علامات الطلاب نستخدم التعليمة:

```
select sum(studentGrade) from grades
```

انتبه:

لا يمكن استخدام التابع SUM على أنواع حقول ذات أنماط غير مناسبة كونه يعتمد عملية الجمع الحسابي، أي لا يمكننا، على سبيل المثال، استخدام تابع SUM مع حقل من نمط سلسلة محارف.

يحسب التابع SUM مجموع قيم حقل معين:

- يُستخدم الخيار All عندما نريد الحصول على مجموع قيم حقل معين بما فيها القيم المكررة. يُعتبر هذا الخيار هو الخيار التلقائي في حال عدم تحديد أي من الخيارين Distinct أو All.
- يستخدم الخيار Distinct عندما نريد الحصول على مجموع القيم حقل معين مع استبعاد أي تكرار في القيم.

تجميع البيانات 1

عندما نتكلم عن التوابع التجميعية فلابد لنا أن نتساءل: هل نستطيع أن نطبق التوابع التجميعية على مجموعات جزئية من السجلات بدلاً من تطبيقها على كامل السجلات؟

فإذا كان لدينا جدول منتجات (Products) وأردنا حساب مجموع كلف المنتجات (unitCost) التي نحصل عليها من المورد الأول، وحساب مجموع كلف المنتجات التي نحصل عليها من المورد الثاني، وحساب مجموع كلف المنتجات التي نحصل عليها من المورد الثالث، فإننا سنحتاج لكتابة ثلاث تعليمات منفصلة تعتمد على التابع التجميعي SUM كما يلي:

```
Select sum(unitCost) from products where supplier ID= 1;

Select sum(unitCost) from products where supplier ID= 2;

Select sum(unitCost) from products where supplier ID= 3;
```

لكن هل يمكننا أن نصل إلى تركيب شبيه بالتركيب الظاهر في المثال السابق باستخدام تعليمة واحدة؟

تغطي SQL هذا المتطلب بآلية تساعد على تقسيم السجلات إلى مجموعات جزئية وتساعد على تطبيق التوابع التجميعية على كل مجموعة جزئية على حدى.

عندما نتكلم عن التوابع التجميعية فلابد لنا أن نتساءل: هل نستطيع أن نطبق التوابع التجميعية على مجموعات جزئية من السجلات بدلاً من تطبيقها على كامل السجلات؟
فإذا كان لدينا جدول منتجات، وأردنا حساب مجموع كلف المنتجات التي نحصل عليها من المورد الأول، وحساب مجموع كلف المنتجات التي نحصل عليها من المورد الثاني، وحساب مجموع كلف المنتجات التي نحصل عليها من المورد الثالث، فإننا سنحتاج لكتابة ثلاث تعليمات منفصلة تعتمد على التابع التجميعي SUM، لكن هل يمكننا أن نصل إلى نفس النتيجة بتعليمة واحدة فقط؟

تغطي SQL هذا المتطلب بآلية تساعد على تقسيم السجلات إلى مجموعات جزئية وتساعد على تطبيق التوابع التجميعية على كل مجموعة جزئية على حدى.

تجميع البيانات 2

لتجميع البيانات في SQL نستخدم تعليمة Group by والتي تأخذ الصيغة التالية:

```
Select columnA, aggFunc (aggFuncSpec) from table
where whereSpec
Group by columnA
```

مثال:

إذا كان لدينا جدول Sales يحتوي أسماء المنتجات ProductName وكمية البيع في مراكز مختلفة Quantity، يمكننا كتابة التعليمة التي تعطي الكمية المباعة من كل منتج في جميع المراكز بعد تاريخ معين.

```
Select productName, sum (quantity) from sales
where saleDate > 'May 2,2002'
Group by productName
```

لتجميع البيانات في SQL نستخدم تعليمة Group by.

الكلمة المفتاحية Having

- عندما نفكر في وضع شرط ما على استعلام معين، يرد إلى ذهننا استخدام الكلمة المفتاحية where مع شرط مناسب.

```
Select field_name from table_name where condition
```

ولكن هذا الحل البديهي يكون قاصراً في بعض الحالات مثل الحالة التي يكون فيها أحد عناصر الشرط تابعاً تجميعياً.

- نستخدم الكلمة المفتاحية Having في حال أردنا أن نضع شرطاً على استعلام ما، بحيث يكون أحد عناصرالشرط تابعاً تجميعياً.

- نستعمل Having ضمن الصيغة التالية:

```
Select columnA, aggFunc (aggFuncSpec) from table
where whereSpec
Group by columnA
Having filterCondition
```

انتبه: لا تلغي الكلمة المفتاحية Having دور where بل يمكن استخدامهما معاً في صيغة واحدة كما يظهر في الصيغة السابقة

مثال:

إذا كان لدينا الجدول StudentsGrade الحاوي على أرقام الطلاب (StudentNumber) وعلى علاماتهم (grade) في مختلف المواد، وأردنا معرفة أي من الطلاب قد حصل على معدل جيد جداً (أكبر من 70) أو سيئ (أصغر من 50)، سيخطر لنا استخدام التعبير التالي:

```
Select studentNumber, Avg(grade) as averageMark from studentGrades
Where avg(grade)>70 or avg(grade)<50
Group by studentNumber
```

إن الحل السابق خاطئ، لأن where تقوم بعملية الترشيح قبل تنفيذ التابع التجميعي (avg) وليس بعده، اذا نحن بحاجة لاستخدام Having لحلّ هذه المسألة. يكون الحل كما يلي:

```
Select studentNumber, Avg(grade) as averageMark from studentGrades
Group by studentNumber
Having avg(grade)>70 or avg(grade)<50
```

عندما نفكر في وضع شرط ما على استعلام معين، يرد إلى ذهننا استخدام الكلمة المفتاحية where مع شرط مناسب.

ولكن هذا الحل البديهي يكون قاصراً في بعض الحالات مثل الحالة التي يكون فيها أحد عناصر الشرط تابعاً تجميعياً

عندها يأتي دور الكلمة المفتاحية having في حال أردنا أن نضع شرطاً على استعلام ما، بحيث يكون أحد عناصرالشرط تابعاً تجميعياً.

عموماً، لا بد من الإشارة إلى أن الكلمة المفتاحية Having لا تلغي دور الكلمة المفتاحية where بل يمكن استخدامهما معاً في صيغة واحدة.

التعبير Top N

يُستخدم التعبير (Top N) كمرافق للتوابع التجميعية ولكن استخدامه لا يقتصر عليها فقط.
يُعيد هذا التعبير أول N سجل من نتيجة الاستعلام.
يأخذ هذا التعبير الصيغة:

```
Select top N field1, field2 … from table_name
```

مثال:

لنفرض أن لدينا جدول StudentsGrade التالي الذي يحتوي على أسماء الطلاب (StudentName) وعلاماتهم (StudentMark) في جميع المواد، ولنفرض أننا أردنا معرفة أسماء الطلاب الخمسة الأوائل مُرتبة، بحسب معدلاتهم، ترتيباً تنازلياً (معدل الطالب هوالمتوسط الحسابي لجميع علاماته).

studentName	studentMark	subject
ahmad	15	math
adel	22	math
ahmad	26	history
…	…	…

نستخدم الصيغة:

```
Select top 5 studentName, avg(studentMark)
From studentsGrades
Group by studentName
order by avg(studentMark) DESC
```

يُستخدم التعبير (Top N) كمرافق للتوابع التجميعية ولكن استخدامه لا يقتصر عليها فقط. ويُعيد هذا التعبير أول N سجل من نتيجة الاستعلام.

استخدام التعبير Top N في مختلف أنظمة إدارة قواعد المعطيات

تختلف الصيغة الخاصة بالتعبير (Top N)، من نظام إدارة قواعد بيانات إلى آخر.

- ففي Mysql تكون الصيغة على الشكل التالي:

```
Select field1, field2 from table_name
Limit 0,N
```

حيث تشير الكلمة المفتاحية limit إلى مجال أرقام السجلات التي نريد الحصول عليها (سجل البداية 0 وسجل النهاية N) من مجموعة السجلات التي تعيدها تعليمة Select.

- أما في قواعد بيانات DB2 فتكون الصيغة على الشكل التالي:

```
Select field1, field2 from table_name
Fetch first N rows only
```

حيث يحدد المعامل N في التعبير "fetch first N rows only" عدد السجلات التي نريد الحصول عليها وذلك ابتداءاً من السجل الأول.

- وتعتمد الصيغة في قواعد بيانات Oracle على الكلمة المفتاحية rowNum وهي عبارة عن قيمة تلقائية يولدها نظام Oracle وتحدد ترتيب السجلات التي نريد الحصول عليها ضمن مجموعة السجلات التي تعيدها تعليمة Select. فتكون الصيغة على الشكل:

```
Select field1, field2 where rowNum<= N
```

مثال:

في حال أردنا الحصول على الأرقام الثلاثة الأولى التي سجلت أكبر عدد من الاتصالات لدى شركة هواتف محمولة وذلك اعتباراً من سجل اتصالات الزبائن (callLog) المؤلف من حقول: إسم المستخدم (userName)، ورقم هاتف المستخدم (phoneNumber). ويفرض أن نظام إدارة قواعد المعطيات المُستخدم هو نظام Oracle، نستخدم الصيغة:

```
Select phoneNumber, count(userName) from callLog
Group by phoneNumber
Order by count(userName)
Where rowNum<= 3
```

تختلف الصيغة الخاصة بالتعبير (Top N)، من نظام إدارة قواعد بيانات إلى آخر.

ففي Mysql نستخدم الكلمة المفتاحية limit التي تشير إلى مجال أرقام السجلات التي نريد الحصول عليهاK من مجموعة السجلات التي تعيدها تعليمة Select.

أما في قواعد بيانات DB2 فنستخدم التعبير "fetch first N rows only" الذي يشير إلى عدد السجلات التي نريد الحصول عليها وذلك ابتداءاً من السجل الأول.

وتعتمد الصيغة في قواعد بيانات Oracle على الكلمة المفتاحية rowNum وهي عبارة عن قيمة تلقائية يولدها نظام Oracle وتحدد ترتيب السجلات التي نريد الحصول عليها ضمن مجموعة السجلات التي تعيدها تعليمة Select.

مسألة:

تحتفظ شركة متعددة الفروع بسجل خاص بالمهام والأعمال التي نفذها موظفو الشركة والدخل الناتج عن كل مهمة من هذه المهام. بفرض أن جدول المعلومات المتوفرة هو جدول المهام tasks التالي:

taskID	taskDate	taskIncom	taskHandler
1	27/1/2003	10000	adel
…	…	…	…

حيث يعبر الحقل taskID عن رقم المهمة

ويعبر الحقل taskDate عن تاريخ المهمة

ويعبر الحقل taskIncom عن الدخل الناتج عن المهمة

ويعبر الحقل taskHandler عن اسم الموظف الذي نفذ المهمة

والمطلوب الحصول على جدول يحدد:

- عدد الأعمال التي قام بها كل موظف.
- الربح الكلي الذي حصلت عليه الشركة من عمل هذا الموظف وذلك خلال العام 2004.

مع ترتيب الجدول على نحو يظهر فيه الموظف الذي نفذ المهمة التي حققت أعلى دخل للمؤسسة، في أول سجل في الجدول.

الحل:

سيظهر في جدول النتيجة معلومات حول اسم الموظف، وعدد المهمات التي أوكلت له، ومجموع الدخول التي حصلت عليها الشركة من عمل الموظف، وقيمة الدخل الأعظمي الذي حصل عليه الموظف من مهامه، مع ترتيب قيم الدخول الأعظمية ترتيباً تنازلياً، فيكون أول اسم في الجدول هو اسم الموظف الذي نفذ المهمة التي حققت أعلى دخل للمؤسسة.

يمكن الحصول على الحل المطلوب بالصيغة التالية:

```
Select taskHandler, count(taskID), sum(taskIncom), max(taskIncom)
From tasks
Group by taskHandler
Where taskDate between 01/01/2004 and 31/12/2004
Order by max(taskIncom) DESC
```

تحتفظ شركة متعددة الفروع بسجل خاص بالمهام والأعمال التي نفذها موظفو الشركة والدخل الناتج عن كل مهمة من هذه المهام. بفرض أن جدول المعلومات المتوفرة هو جدول المهام tasks التالي والمؤلف من الحقل taskID الذي يُعبر عن رقم المهمة، والحقل taskDate الذي يُعبر عن تاريخ المهمة، والحقل taskIncom الذي يُعبر عن الدخل الناتج عن المهمة، والحقل taskHandler الذي يُعبر عن اسم الموظف الذي نفذ المهمة.

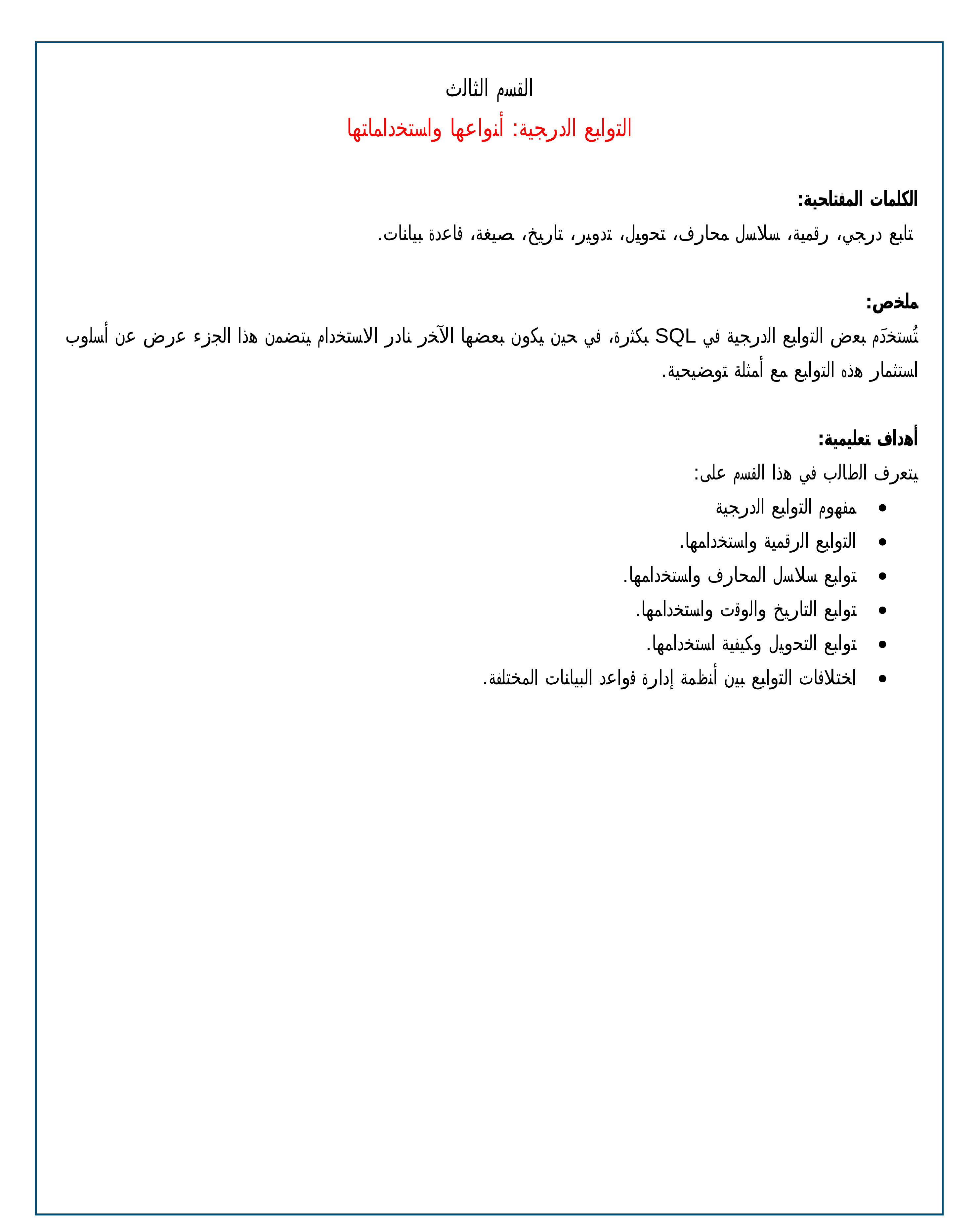
والمطلوب الحصول على جدول يحدد:

- عدد الأعمال التي قام بها كل موظف.
- الربح الكلي الذي حصلت عليه الشركة من عمل هذا الموظف وذلك خلال العام 2004.

مع ترتيب الجدول على نحو يظهر فيه الموظف الذي نفذ المهمة التي حققت أعلى دخل للمؤسسة، في أول سجل في الجدول.

الحل:

سيظهر في جدول النتيجة معلومات حول اسم الموظف، وعدد المهمات التي أوكلت له، ومجموع الدخول التي حصلت عليها الشركة من عمل الموظف، وقيمة الدخل الأعظمي الذي حصل عليه الموظف من مهامه، مع ترتيب قيم الدخول الأعظمية ترتيباً تنازلياً، فيكون أول اسم في الجدول هو اسم الموظف الذي نفذ المهمة التي حققت أعلى دخل للمؤسسة.



القسم الثالث

التوابع الدرجية: أنواعها واستخداماتها

الكلمات المفتاحية:

تابع درجي، رقمية، سلاسل محارف، تحويل، تدوير، تاريخ، صيغة، قاعدة بيانات.

ملخص:

تُستخدَم بعض التوابع الدرجية في SQL بكثرة، في حين يكون بعضها الآخر نادر الاستخدام يتضمن هذا الجزء عرض عن أسلوب استثمار هذه التوابع مع أمثلة توضيحية.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- مفهوم التوابع الدرجية
- التوابع الرقمية واستخدامها.
- توابع سلاسل المحارف واستخدامها.
- توابع التاريخ والوقت واستخدامها.
- توابع التحويل وكيفية استخدامها.
- اختلافات التوابع بين أنظمة إدارة قواعد البيانات المختلفة.

توابع SQL الدرجية

- سبق وعرفنا التوابع الدرجية في SQL على أنها التوابع التي تأخذ مُعاملاً وحيداً وتُعيد قيمة وحيدة.
- تختلف التوابع الدرجية من نظام إدارة قواعد بيانات إلى آخر.
- سنركز في دراستنا على أهم التوابع الدرجية التي حددها المعيار SQL99 إضافة إلى بعض التوابع الأخرى الهامة.
- نصنف التوابع الدرجية إلى أربعة أصناف أساسية هي التالية:

التوابع الرقمية	وهي التوابع الخاصة بالعمليات على الأرقام، مثل تابع التقريب إلى أقرب فاصلة عشرية
توابع سلاسل المحارف	وهي التوابع الخاصة بالعمليات على سلاسل المحارف، مثل تابع تحديد طول سلسلة محرفية
توابع التاريخ والوقت	وهي التوابع الخاصة بالعمليات على التاريخ والوقت، مثل تابع حساب الزمن الفاصل بين تاريخين
توابع التحويل	هي التوابع الخاصة بعملية تحويل مُعامل الدخل، من نمط بيانات إلى آخر.

انتبه: قد لا تأخذ بعض التوابع الدرجية أية مُعاملات مباشرةً ، أي أنها قد تعتمد على مُعاملات مصدرها قاعدة البيانات نفسها.

توابع SQL الدرجية

سبق وعرفنا التوابع الدرجية في SQL على أنها التوابع التي تأخذ مُعاملاً وحيداً وتُعيد قيمة وحيدة.
إلا أن التوابع الدرجية كثيرة وتختلف من نظام إدارة قواعد بيانات إلى آخر، لذا سنركز في دراستنا على أهم التوابع الدرجية التي حددها المعيار SQL-99 إضافة إلى بعض التوابع الأخرى الهامة.

نصنف التوابع الدرجية إلى أربعة أصناف أساسية هي التالية:

- **التوابع الرقمية:** وهي التوابع الخاصة بالعمليات على الأرقام، مثل تابع التقريب إلى أقرب فاصلة عشرية
- **توابع سلاسل المحارف:** وهي التوابع الخاصة بالعمليات على سلاسل المحارف، مثل تابع تحديد طول سلسلة محرفية
- **توابع التاريخ والوقت:** وهي التوابع الخاصة بالعمليات على التاريخ والوقت، مثل تابع حساب الزمن الفاصل بين تاريخين.
- **توابع التحويل:** هي التوابع الخاصة بعملية تحويل مُعامل الدخل من نمط بيانات إلى نمط آخر.

التوابع الرقمية

هي التوابع الخاصة بالعمليات على القيم الرقمية ومن أهمها التوابع التالية:

التابع	استخدامه
Floor( )	يُقرب مُعامل الدخل إلى أقرب عدد صحيح أصغر من معامل الدخل
Ceiling( )	يُقرب مُعامل الدخل إلى أقرب عدد صحيح أكبر من مُعامل الدخل
Round( )	يُقرب مُعامل الدخل ذو الفاصلة العشرية إلى أقرب عدد صحيح أو عدد حقيقي بدقة محددة
Abs( )	يعيد القيمة المطلقة لمُعامل الدخل.
Sin( ), Cos( ), Tan( ), Atan( ),	تحسب قيم ظل، تظل، جب، تجب، الخاصة بمُعامل الدخل.
SQRT( )	يُعيد قيمة الجذر التربيعي لمُعامل الدخل
RAND( )	يُعيد رقم عشوائي بين 0 و1 ويستخدم مُعامل الدخل كأساس لتوليد الرقم العشوائي.

التوابع الرقمية وهي التوابع الدرجية الخاصة بالعمليات على القيم الرقمية ومن أهمها التوابع التالية:

- التابع Floor: وهو التابع الذي يُقرب مُعامل الدخل إلى أقرب عدد صحيح أصغر من معامل الدخل.
- التابع Ceiling: وهو التابع الذي يُقرب مُعامل الدخل إلى أقرب عدد صحيح أكبر من مُعامل الدخل.
- التابع Round: وهو التابع الذي يُقرب مُعامل الدخل ذو الفاصلة العشرية إلى أقرب عدد صحيح أو عدد حقيقي بدقة محددة.
- التابع Abs: وهو التابع الذي يعيد القيمة المطلقة لمُعامل الدخل.
- التوابع Sin, Cos, Tan, Atan: وهي التوابع التي تحسب قيم ظل، تظل، جب، تجب الزاوية التي نأخذها كمُعامل دخل.
- التابع SQRT: وهو التابع الذي يُعيد قيمة الجذر التربيعي لمُعامل الدخل.
- التابع RAND: وهو التابع الذي يُعيد رقم عشوائي بين 0 و1 ويستخدم مُعامل الدخل كأساس لتوليد الرقم العشوائي.

التوابع الرقمية - أمثلة

نتناول فيما يلي أمثلة عن استخدام التوابع الدرجية الرقميّة في أنظمة إدارة قواعد البيانات المختلفة:

- **التابع Floor:**

نستخدم في هذا المثال تابع التدوير Floor لتدوير علامات الطلاب إلى الرقم الصحيح بإزالة الجزء ذو الفاصلة العشرية.

```
Select floor(studentMark) from marks
```

فإذا كان لدينا طالب علامته: 66.5، أو 66.2، أو 66.7، ستظهرعلامته ضمن نتيجة هذا الاستعلام: 66

- **التابع Ceiling:**

إذا كان المطلوب تدوير علامات الطلاب إلى الرقم الصحيح الأعلى، نستخدم التابع Ceiling كما في الصيغة التالية:

```
Select ceiling(studentMark) from marks
```

فإذا كان لدينا طالب علامته: 66.5، أو 66.2، أو 66.7، ستظهرعلامته ضمن نتيجة هذا الاستعلام: 67

- **التابع Round:**

إذا كان المطلوب تدوير عدد ما إلى رقم بعدد خانات عشرية محدد نستطيع استخدام التابع Round كما في الصيغة التالية:

```
Select round(studentMark, 1) from marks
```

تظهر قيم الخرج كقيم ذات خانة عشرية واحدة. فإذا كانت علامة طالب 66 ستظهر القيمة ضمن نتيجة الاستعلام 66.0 وإذا كانت 66.55 ستصبح 66.5

انتبه:

لا يتعرف نظام إدارة قواعد البيانات Access على التوابع الدرجية Floor وceiling. لكن يمكن الوصول إلى نفس النتائج باستخدام تابع INT للاستعاضة عن تابع Floor. فلو كان المطلوب تدوير علامات الطلاب بإزالة الجزء العشري والإبقاء على العدد الصحيح نستخدم:

```
Select Int(studentMark) from marks
```

وللاستعاضة عن Ceiling نستخدم التابع Round. فلو كان المطلوب تدوير علامات الطلاب إلى العدد الصحيح الأكبر نستخدم الصيغة:

```
Select Round(studentMark+0.5,0) from marks
```

أما في Oracle فنستخدم التابع Ceil عوضاً عن Ceiling

في هذه الشريحة نوضح بالأمثلة عمل التوابع الدرجية الرقمية الخاصة بتدوير القيم العددية:

إذ نستخدم تابع التدوير Floor لتدوير علامات الطلاب إلى الرقم الصحيح، وذلك بإزالة الجزء ذو الفاصلة العشرية. فإذا كان لدينا طالب علامته: 66.5، أو 66.2، أو 66.7، ستظهرعلامته عند استخدام التابع Floor: 66

أما إذا كان المطلوب تدوير علامات الطلاب إلى الرقم الصحيح الأعلى، فنستخدم عندها التابع Ceiling. فإذا كان لدينا طالب علامته: 66.5، أو 66.2، أو 66.7، ستظهرعلامته عند استخدام التابع Ceiling: 67

وإذا كان المطلوب تدوير عدد ما إلى رقم بعدد خانات عشرية محدد، نستطيع استخدام التابع **Round**، حيث تظهر قيم الخرج كقيم ذات خانة عشرية واحدة. فإذا كانت علامة طالب 66، ستظهر القيمة عند استخدام التابع **Round**، 66.0، وإذا كانت 66.55، ستظهر 66.5.

التوابع الرقمية - أمثلة

تشمل التوابع الدرجية الرقمية، كما أسلفنا، توابع التدوير إضافة إلى توابع أخرى مثل تابع القيمة المطلقة، وتابع الجذر التربيعي، وتابع القيمة العشوائية وتوابع المثلثات.

فيما يلي مجموعة من الأمثلة التي تغطي التوابع الدرجية الرقمية الأخرى:

- **التابع ABS:**

إذا أردنا حساب القيمة المطلقة لحقل height يشير إلى ارتفاع نقاط جغرافية عن سطح البحر من الجدول geoTable، وذلك لتحديد الارتفاع الأكبر بينها، يمكننا استخدام الاستعلام:

```
Select Max(abs(height)) from geoTable
```

يحسب التابع abs القيمة المطلقة لكل ارتفاع، ويستخرج التابع التجميعي max الإرتفاع الأعلى.

- **التوابع Sin, Cos, Tan...**

إذا أردنا حساب قيمة، جب أوتجب أوظل مجموعة من الزوايا الواردة في الحقل Angle من الجدول Angles يكفي كتابة الاستعلام:

```
Select sin(angle), cos(angle), tan(angle) from Angles
```

- **التابع Rand:**

يُستخدَم التابع Rand في توليد رقم عشوائي بين 0 و1، فإذا كان المطلوب توليد رقم عشوائي مستخدمين كأساس للتوليد الأرقام الواردة في حقل seed من الجدول Numbers أمكننا كتابة الاستعلام:

```
Select rand(seed) from numbers
```

- **التابع SQRT:**

يحسب التابع SQRT الجذر التربيعي لقيمة معينة، إذ تعطي نتيجة الاستعلام التالي، الرقم 3:

```
select sqrt(9)
```

انتبه:

تكون صيغة تابع توليد رقم عشوائي بين 0 و1 في MS-Access rnd وليس rand.

تشمل التوابع الدرجية الرقمية، كما أسلفنا، توابع التدوير إضافة إلى توابع أخرى مثل تابع القيمة المطلقة، وتابع الجذر التربيعي، وتابع القيمة العشوائية وتوابع الزوايا.

فإذا كنا نريد حساب القيمة المطلقة لحقل height يشير إلى ارتفاع نقاط جغرافية عن سطح البحر من الجدول geoTable، وذلك بهدف تحديد الارتفاع الأعلى بينها، يمكننا تطبيق التابع abs على كل ارتفاع مع استخدام التابع التجميعي max لحساب الإرتفاع الأعلى. وإذا أردنا حساب قيمة، جب أوتجب أوظل مجموعة من الزوايا يمكننا استخدام التوابع Sin, Cos, Tan

أما التابع Rand، فيُستخدم في توليد رقم عشوائي بين 0 و1. كما يحسب التابع SQRT الجذر التربيعي لقيمة معينة.

توابع سلاسل المحارف

هي التوابع الخاصة بالعمليات على سلاسل المحارف، ومن أهمها التوابع التالية:

التابع	استخدامه
Left()	يُعيد جزء من السلسلة يبتدئ من بدايتها حتى عدد محدد من المحارف
Right()	يُعيد جزء من السلسلة يبتدئ من نهايتها حتى عدد محدد من المحارف
Substr()	يُعيد جزء من السلسلة يبتدئ من موقع محدد فيها وبطول عدد محدد من المحارف
Length()	يُعيد طول السلسلة المحرفية
Concat()	يُستخدم لدمج أكثر من سلسلة محرفية
Lower() / Upper()	يحول جميع محارف السلسلة إلى أحرف كبيرة أو صغيرة
Trim()	يلغي الفراغات من بداية ونهاية السلسلة المحرفية
Instr()	يُستخدم لتحديد موقع سلسلة جزئية ضمن سلسلة رئيسية

توابع سلاسل المحارف هي التوابع الدرجية الخاصة بالعمليات على السلاسل المحرفية، ومن أهمها التوابع التالية:

Left()	وهو التابع الذي يُعيد جزء من السلسلة، يبتدئ من بدايتها حتى عدد محدد من المحارف
Right()	وهو التابع الذي يُعيد جزء من السلسلة، يبتدئ من نهايتها حتى عدد محدد من المحارف
Substr()	وهو التابع الذي يُعيد جزء من السلسلة، يبتدئ من موقع محدد فيها وبطول عدد محدد من المحارف
Length()	وهو التابع الذي يُعيد طول السلسلة المحرفية
Concat()	وهو التابع الذي يُستخدم لدمج أكثر من سلسلة محرفية
Lower() / Upper()	وهو التابع الذي يحول جميع محارف السلسلة إلى أحرف كبيرة أو صغيرة
Trim()	وهو التابع الذي يلغي الفراغات من بداية ونهاية السلسلة المحرفية
Instr()	وهو التابع الذي يُستخدم لتحديد موقع سلسلة جزئية ضمن سلسلة رئيسية

توابع سلاسل المحارف - أمثلة

فيما يلي أمثلة عن أهم توابع سلاسل المحارف:

- **التابعان Left وRight:**

يُعيد التابعان Left و Right جزء من سلسلة المحارف مؤلف من عدد محدد من المحارف ابتداءاً من يسار السلسلة في حالة التابع Left، ومن يمين السلسلة في حالة التابع Right. فإذا أردنا كتابة الاستعلام الذي يعيد أول 50 محرفاً من قيم الحقل Title في الجدول News يمكننا استخدام الصيغة التالية:

```
Select left(title, 50) from News
```

- **التابع Substr:**

يُعيد التابع Substr جزء من سلسلة محارف، ابتداءاً من موقع محدد في تلك السلسلة، وبطول عدد محدد من المحارف. فإذا أردنا كتابة الاستعلام الذي يعيد سلسلة محارف بطول 5 محارف اعتباراً من قيم الحقل Title مبتدئاً من المحرف رقم 10، نكتب الصيغة التالية:

```
Select substr(title, 10, 5) from News
```

- **التابع Length:**

يحسب التابع Length طول سلسلة محارف معينة. فإذا أردنا كتابة الاستعلام الذي يعيد أطوال سلاسل المحارف الخاصة بالحقل Title نستخدم الصيغة:

```
Select length(title) from News
```

- **التابع Concat:**

يدمج التابع Concat أكثر من سلسلة محرفية. فلإعادة قائمة بسلاسل محرفية تمثل حاصل دمج السلاسل المحرفية الخاصة بالحقل Title، مع السلاسل المحرفية الخاصة بالحقل Details في الجدول News، نكتب الصيغة:

```
Select concat(title, details) from News
```

انتبه:

- لا يستخدم نظام إدارة قواعد البيانات Oracle التوابع Left وRight، ويتم الاستعاضة عنها باستخدام التابع Substr.
- يستخدم نظام Ms-Access ونظام SQL-Server الصيغة Len عوضاً عن التابع Length، لحساب طول السلسلة المحرفية.
- يستخدم كلاً من نظام Mysql ونظام SQL-Server التابع Substring عوضاً عن التابع Substr، لإعادة جزء من سلسلة محارف.

ذكرنا أن توابع سلاسل المحارف هي التوابع الخاصة بالعمليات على سلاسل المحارف. إذ يُعيد التابعان Left و Right جزء من سلسلة المحارف مؤلف من عدد محدد من المحارف، ابتداءاً من يسار السلسلة في حالة التابع Left، ومن يمين السلسلة في حالة التابع Right. كما يُعيد التابع Substr جزء من سلسلة محارف ابتداءاً من موقع محدد في تلك السلسلة وبطول عدد محدد من المحارف. ويحسب التابع Length بحساب طول سلسلة محارف معينة. كما يدمج التابع Concat أكثر من سلسلة محرفية ضمن سلسلة واحدة.

توابع سلاسل المحارف - أمثلة

- **التابعان Upper وLower:**

يحول التابعان Upper وLower جميع محارف السلسلة إلى أحرف كبيرة أو صغيرة (فقط للمحارف A-Z). فإذا أردنا تحويل السلاسل المحرفية في الحقل Title إلى سلاسل مؤلفة من أحرف كبيرة، نستخدم الاستعلام:

```
Select upper(title) from News;
```

- **التابع Trim:**

يزيل التابع Trim الفراغات من بداية ونهاية السلسلة المحرفية. فإذا أردنا الحصول على السلاسل المحرفية في الحقل Title بعد التخلص من الفراغات، في بداية ونهاية سلاسل المحارف تلك، نستخدم الصيغة:

```
Select trim(title) from News;
```

- **التابع Instr:**

يحدد التابع Instr موقع أول ظهور لسلسلة جزئية في سلسلة رئيسية. يعيد التابع القيمة 0 في حال عدم ايجاد السلسلة. فإذا كان المطلوب مثلاً كتابة استعلام يعيد موقع أول ظهور للسلسلة الجزئية 'Test' ضمن سلاسل الحقل Title من جدول News، نستخدم الصيغة:

```
Select Instr(title, 'Test') from News;
```

انتبه:

- يستخدم Ms-Access التابعين Ucase وLcase لتحويل جميع محارف السلسلة إلى أحرف كبيرة أو صغيرة عوضاً عن التابعين Upper وLower.
- يستخدم كلاً من SQL-Server وCharindex وDB2 التابع Posstr لتحديد موقع أول ظهور لسلسلة جزئية في سلسلة رئيسية عوضاً عن التابع Instr.

يحول التابعان Upper وLower جميع محارف السلسلة إلى أحرف كبيرة أو صغيرة (فقط للمحارف A-Z). ويزيل التابع Trim الفراغات من بداية ونهاية السلسلة المحرفية. كما يحدد التابع Instr موقع أول ظهور لسلسلة جزئية في سلسلة رئيسية، حيث يعيد التابع القيمة 0 في حال عدم ايجاد السلسلة.

توابع التاريخ والوقت

هي التوابع الخاصة بالعمليات على التاريخ والوقت ومن أهمها التوابع التالية:

التابع	استخدامه
DateDiff()	يُعيد الفرق بين تاريخين
GetDate()	يُعيد السنة، والتاريخ، واليوم، والساعة، والدقيقة، والثانية، وأجزاء الثانية
CURRENT_DATE	يعيد التاريخ الحالي الخاص بنظام إدارة قاعدة البيانات
CURRENT_TIME	يُعيد التوقيت الخاص بنظام إدارة قاعدة البيانات
CURRENT_TIMESTAMP	يُعيد التاريخ والتوقيت الخاصين بنظام إدارة قاعدة البيانات

انتبه:

توجد الكثير من التوابع التي تعيد التاريخ والوقت مثل Days, Hours, Minutes, Seconds ولكن يُفَضل استخدام التابع DateDiff كبديل لها.

توابع التاريخ والتوقيت هي التوابع الخاصة بالعمليات على التاريخ والوقت ومن أهمها التوابع التالية:

التابع	استخدامه
DateDiff()	وهو التابع الذي يُعيد الفرق بين تاريخين
GetDate()	وهو التابع الذي يُعيد السنة، والتاريخ، واليوم، والساعة، والدقيقة، والثانية، وأجزاء الثانية
CURRENT_DATE	وهو التابع الذي يُعيد التاريخ الحالي الخاص بنظام إدارة قاعدة البيانات
CURRENT_TIME	وهو التابع الذي يُعيد التوقيت الخاص بنظام إدارة قاعدة البيانات
CURRENT_TIMESTAMP	وهو التابع الذي يُعيد التاريخ والتوقيت الخاصين بنظام إدارة قاعدة البيانات

توابع التاريخ والوقت

من أهم توابع التاريخ والوقت وأكثرها استخداماً التوابع التالية:

- **التابع GetDate:**

الذي يُعيد التاريخ الحالي متضمناً السنة، والشهر، واليوم، والساعة، والدقيقة، والثانية وجزء الثانية. فلإظهار التاريخ الحالي يمكننا استخدام الصيغة:

```
getDate ()
```

حيث تظهر النتيجة كما يلي: 2003-12-06 04:50:32.28

- **التابع DateDiff:**

الذي يُعيد الفرق بين تاريخين. فإذا أردنا إظهار عدد الأيام الفاصلة بين التاريخ الحالي والتواريخ من قيم الحقل RegistrationDate في الجدول RegistrationInfo نستخدم الصيغة:

```
Select dateDiff(dd, RegistrationDate, getDate()) from
registrationInfo
```

استخدمنا المُعامل dd لتحديد صيغة خرج التابع حيث أننا نريد إظهار الفرق بين التاريخين بالأيام. اضغط هنا لإظهار بقية المُعاملات التي يمكن أن تحل محل المُعامل dd.

انتبه:

- يستخدم Ms-Access التابع Date عوضاً عن التابع getDate.
- يسمح Oracle بتنفيذ عملية طرح مباشر لتاريخين.

من أهم توابع التاريخ والوقت وأكثرها استخداماً، التابع GetDate، الذي يُعيد التاريخ الحالي متضمناً السنة، والشهر، واليوم، والساعة، والدقيقة، والثانية وجزء الثانية. بالإضافة إلى التابع DateDiff، الذي يُعيد الفرق بين تاريخين.

توابع التاريخ والوقت - أمثلة

- **التابع CURRENT_DATE**

يُعيد هذا التابع قيمة التاريخ الخاص بنظام إدارة قاعدة البيانات (يُعتبر أحد متحولات نظام إدارة قواعد البيانات وهو ليس تابع فعلي). فإذا أردنا إظهار التاريخ الحالي نستخدم الصيغة:

```
Select CURRENT_DATE as myDate
```

- **التابعCURRENT_TIME**

يُعيد هذا التابع قيمة التوقيت الحالي الخاص بنظام إدارة قاعدة البيانات (يُعتبر أحد متحولات نظام إدارة قواعد البيانات وهو ليس تابع فعلي). فإذا أردنا إظهار التوقيت الحالي نستخدم الصيغة:

```
Select CURRENT_TIME as myTime
```

- **التابع CURRENT_TIMESTAMP**

يُعيد هذا التابع قيمة قيمة التاريخ والتوقيت الخاص بنظام إدارة قاعدة البيانات (يُعتبر أحد متحولات نظام إدارة قواعد البيانات وهو ليس تابع فعلي)، فالقيمة التي يعيدها التابع CURRENT_TIMESTAMP مشابهة تماماً للقيمة التي يعيدها التابع GetDate(). فإذا أردنا إظهار عدد الأيام الفاصلة بين التاريخ الحالي والتواريخ المُخزنة في الحقل RegistrationDate من الجدول RegistrationInfo نستخدم الصيغة:

```
Select dateDiff(dd, RegistrationDate, CURRENT_TIMESTAMP) from
registrationInfo
```

يُعيد **التابع** CURRENT_DATE قيمة التاريخ الخاص بنظام إدارة قاعدة البيانات.

كما يُعيد التابع CURRENT_TIME قيمة التوقيت الحالي الخاص بنظام إدارة قاعدة البيانات.

ويُعيد التابع CURRENT_TIMESTAMP قيمة التاريخ والتوقيت الخاص بنظام إدارة قاعدة البيانات فالقيمة التي يعيدها التابع CURRENT_TIMESTAMP مشابهة تماماً للقيمة التي يعيدها التابع ()GetDate.

توابع التحويل

هي التوابع الخاصة بالتحويل من نمط بيانات إلى نمط آخر ومن أهمها التوابع التالية:

التابع	استخدامه
Str()	يُحول قيمة الدخل العددية إلى سلسلة محرفية.
To_Number()	يُحول سلسلة المحارف المارة كمُعامل دخل إلى عدد.
Cast()	يُحول قيمة الدخل إلى قيمة من أي نمط آخر من البيانات
Convert()	يُحول قيمة الدخل إلى قيمة من أي نمط آخر من البيانات

انتبه:

قد تلاحظ أن بعض توابع التحويل تندرج ضمن أصناف التوابع السابقة، كالتوابع الرقمية أو توابع السلاسل، لكننا فضلنا فصلها في تصنيف منفصل.

توابع التحويل وهي التوابع الخاصة بالتحويل من نوع بيانات إلى نوع آخر ومن أهمها التوابع التالية:

- التابع Str الذي يُحول قيمة الدخل العددية إلى سلسلة محرفية.
- التابع To_Number الذي يُحول سلسلة المحارف المارة كمُعامل دخل إلى عدد.
- التابع Cast الذي يُحول قيمة الدخل إلى قيمة من أي نمط آخر من البيانات
- لتابع Convert: الذي يُحول قيمة الدخل إلى قيمة من أي نمط آخر من البيانات

توابع التحويل - أمثلة

من أهمها التوابع التالية:

- **التابع STR:**

الذي يُحول قيمة الدخل العددية إلى سلسلة محرفية. يكون لهذا التابع الصيغة التالية:

```
STR (Float, Length, Precision)
```

فإذا أردنا مثلاً تحويل العدد 53.45 إلى سلسلة محارف نقوم باستخدام الصيغة:

```
STR(53.45 , 5 , 2)
```

حيث يمثل العدد 5 الطول الكامل للسلسلة المحرفية المُعادة، ويمثل العدد 2، عدد المراتب العشرية.

انتبه: في حال كانت قيمة المعامل Length غير كافية لتتسع للسلسلة المحرفية الناتجة، ستعيد STR سلسلة محرفية ممتلئة بالمحرف '*'

- **التابع To_Number:**

الذي يُحول سلسلة المحارف المارة كمُعامل دخل إلى عدد. يأخذ التابع معاملين، أحدهما السلسلة المحرفية المراد تحويلها إلى رقم، والآخر هو تنسيق يساعد التعليمة على فهم وتحويل السلسلة المحرفية. فإذا أردنا تحويل السلسلة المحرفية '$3,15.2' إلى رقم نستخدم الصيغة:

```
To_Number('$3,15.2' , '$9,99.9')
```

حيث أعطينا المعامل الخاص بتوضيح صيغة السلسلة المحرفية المراد تحويلها، القيمة '$9,99.9' لمساعدة التابع على المقارنة مع هذا المعامل، وتحديد الجزء الذي سوف يتم تحويله.

انتبه:

- لايدعم SQL-Server التابع To_Number بل يستخدم عوضاً عنه التابع Cast
- نستخدم في DB2 التوابع INT, Float, DEC عوضاً عن التابع To_Number
- نضيف في MySql، العدد 0 إلى القيمة التي نود تحويلها فيتم تحويل الناتج تلقائياً إلى عدد
- نستخدم في Ms-Access، التابع INT بدلاً من التابع To_Number

من أهم توابع التحويل التالية التابع STR الذي يُحول قيمة الدخل العددية إلى سلسلة محرفية، والتابع To_Number الذي يُحول سلسلة المحارف، المارة كمُعامل دخل، إلى عدد. يأخذ التابع الأخير معاملين، أحدهما السلسلة المحرفية المراد تحويلها إلى رقم، والآخر هو تنسيق يساعد التعليمة على فهم وتحويل السلسلة المحرفية.

توابع التحويل (تتمة)

- **التابع Cast:**

الذي يُحول قيمة الدخل إلى قيمة من أي نمط آخر من البيانات. يستخدم التابع Cast الصيغة:

```
Cast(Expression as Data_Type)
```

فإذا أردنا مثلاً تحويل السلسلة المحرفية '4.123' إلى عدد عشري بمرتبتين عشريتين نستخدم الصيغة:

```
Cast('4.123' as Decimal(3,2))
```

وتكون القيمة المعادة هي 4.12

- **التابع Convert:**

يُحول قيمة الدخل إلى قيمة من أي نمط آخر من البيانات. يستخدم التابع Convert الصيغة:

```
Convert(Expression, Data_Type)
```

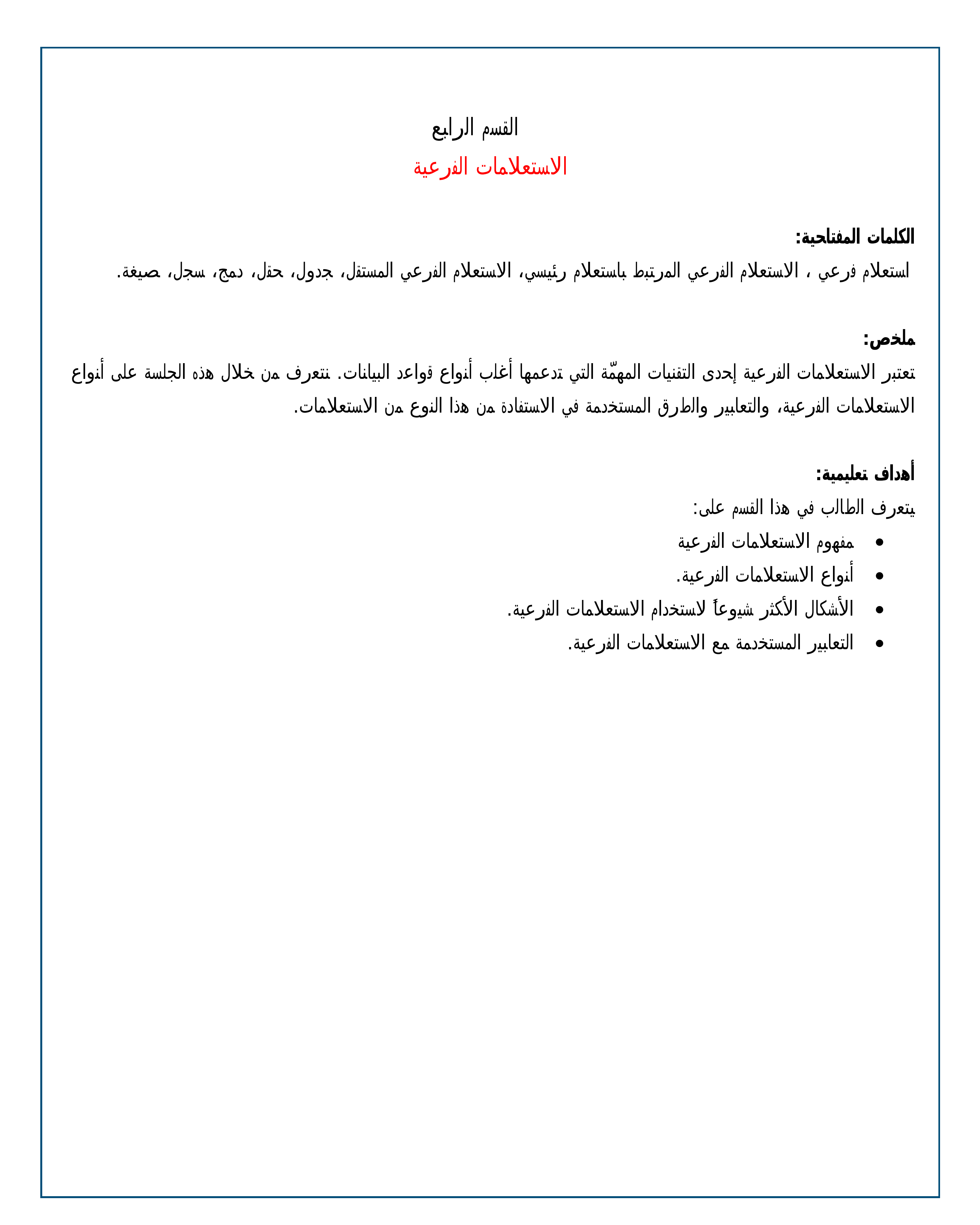
فإذا أردنا تحويل السلسلة المحرفية'5.2' إلى عدد صحيح نستخدم الصيغة:

```
Convert('5.2', integer)
```

انتبه:

- تدعم قواعد البيانات SQL Server, Oracle, DB2, MySql التابع Cast
- تدعم قواعد البيانات MySql وSQL-Server التابع Convert

يحوّل كلاً من التابع Cast والتابع Convert قيمة الدخل إلى قيمة من أي نمط آخر من البيانات



القسم الرابع

الاستعلامات الفرعية

الكلمات المفتاحية:

استعلام فرعي ، الاستعلام الفرعي المرتبط باستعلام رئيسي، الاستعلام الفرعي المستقل، جدول، حقل، دمج، سجل، صيغة.

ملخص:

تعتبر الاستعلامات الفرعية إحدى التقنيات المهمّة التي تدعمها أغلب أنواع قواعد البيانات. نتعرف من خلال هذه الجلسة على أنواع الاستعلامات الفرعية، والتعابير والطرق المستخدمة في الاستفادة من هذا النوع من الاستعلامات.

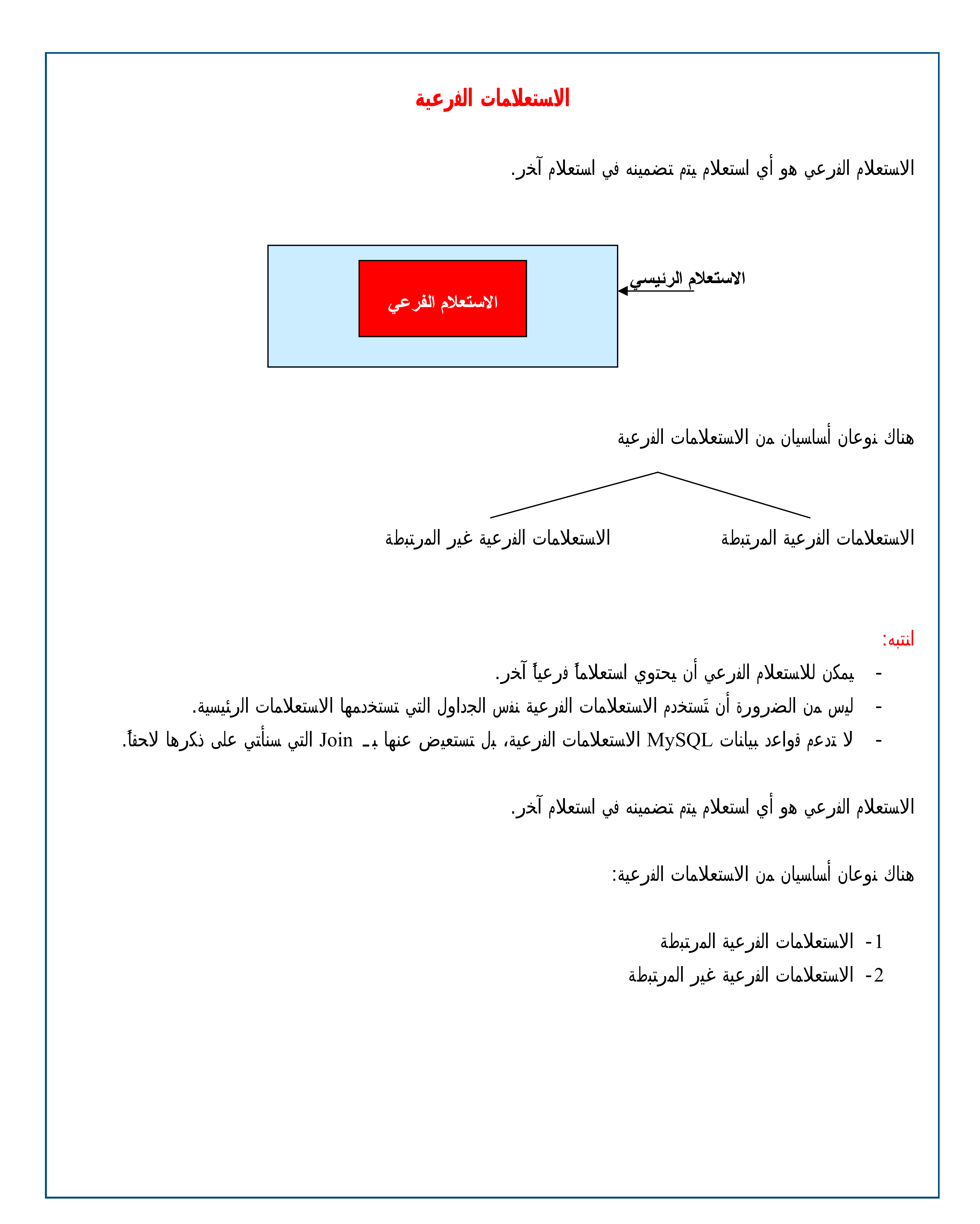
أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- مفهوم الاستعلامات الفرعية
- أنواع الاستعلامات الفرعية.
- الأشكال الأكثر شيوعاً لاستخدام الاستعلامات الفرعية.
- التعابير المستخدمة مع الاستعلامات الفرعية.

الاستعلامات الفرعية

الاستعلام الفرعي هو أي استعلام يتم تضمينه في استعلام آخر.



هناك نوعان أساسيان من الاستعلامات الفرعية

الاستعلامات الفرعية المرتبطة　　　　الاستعلامات الفرعية غير المرتبطة

انتبه:

- يمكن للاستعلام الفرعي أن يحتوي استعلاماً فرعياً آخر.
- ليس من الضرورة أن تَستخدم الاستعلامات الفرعية نفس الجداول التي تستخدمها الاستعلامات الرئيسية.
- لا تدعم قواعد بيانات MySQL الاستعلامات الفرعية، بل تستعيض عنها بـ Join التي سنأتي على ذكرها لاحقاً.

الاستعلام الفرعي هو أي استعلام يتم تضمينه في استعلام آخر.

هناك نوعان أساسيان من الاستعلامات الفرعية:

1- الاستعلامات الفرعية المرتبطة
2- الاستعلامات الفرعية غير المرتبطة

الاستعلامات الفرعية

1- **الاستعلامات الفرعية المرتبطة باستعلام رئيسي:** هي الاستعلامات الفرعية التي تعتمد في عملها على بيانات من استعلامات رئيسية حاوية لها.
في هذا النوع من الاستعلامات الفرعية، يتم تكرار عملية تنفيذ الاستعلام بعدد مرات مساوٍ لعدد السجلات التي يُعيدها الاستعلام الرئيسي.

مثال:

إذا كان لدينا الجدول Customers الحاوي على معلومات الزبائن، والجدول Orders الحاوي على معلومات الطلبيات. لإظهار قائمة باسم كل زبون (customerName) وعدد الطلبيات لكل زبون نكتب الاستعلام:

```
Select customerName,(select count(*) from Orders
where Orders.customerID=Customers.customerID) from Customers;
```

نلاحظ في المثال السابق بأنه سيتم تكرار تنفيذ الاستعلام الفرعي (الظاهر باللون الأحمر) بعدد سجلات الزبائن وأن الاستعلام سيعيد عدد الطلبيات لكل زبون.

2- **الاستعلامات الفرعية المستقلة:** وهي الاستعلامات الفرعية التي تكون مستقلة تماماً عن الاستعلامات الرئيسية الحاوية لها، أي أن الاستعلام الفرعي سيُنَفذ بشكل كامل ويُمرر القيمة أو مجموعة القيم الناتجة إلى الاستعلام الرئيسي.

مثال:

إذا كان لدينا جدول Students يحتوي أسماء الطلاب (studentName) وأرقامهم التسلسلية (studentID). وجدول آخر Grades يحتوي علامات الطلاب (grade) وأرقامهم التسلسلية (studentID). لإعادة لائحة بأسماء الطلاب الناجحين فقط نكتب الاستعلام:

```
Select studentName from Students where Students.studentID in (select
Grades.studentID from Grades where Grades.grade>=50);
```

(اعتُبِرت علامة النجاح 50).

نلاحظ في الاستعلام السابق أنه سيتم تنفيذ الاستعلام الفرعي (باللون الأحمر) مرة واحدة فقط، وبصورة مستقلة عن عدد السجلات في الجدول Students.

انتبه:

- تحتاج الاستعلامات الفرعية المرتبطة باستعلام رئيسي للكثير من الوقت والمعالجة بالمقارنة مع الاستعلامات غير المرتبطة باستعلامها الرئيسي.
- يتطلب العمل بالاستعلامات الفرعية أحياناً استخدام حقول من أكثر من جدول، فاحرص على عدم استخدام أسماء حقول طموحة بل استخدم الصيغة Table_Name.Field_Name.

1- **الاستعلامات الفرعية المرتبطة باستعلام رئيسي**: هي الاستعلامات الفرعية التي تعتمد في عملها على بيانات من استعلامات رئيسية حاوية لها.
   في هذا النوع من الاستعلامات الفرعية، يتم تكرار عملية تنفيذ الاستعلام بعدد مرات مساوٍ لعدد السجلات التي يُعيدها الاستعلام الرئيسي.

2- **الاستعلامات الفرعية المستقلة:** وهي الاستعلامات الفرعية التي تكون مستقلة تماماً عن الاستعلامات الرئيسية الحاوية لها، أي أن الاستعلام الفرعي سيُنَفذ بشكل كامل ويُمرر القيمة أو مجموعة القيم الناتجة إلى الاستعلام الرئيسي.

استعمال الاستعلام الفرعي كعمود من أعمدة الاستعلام الرئيسي

يأخذ هذا الاستعلام الصيغة:

```
Select columnA, (subquery) as columnB from Table_Name;
```

في الصيغة السابقة سوف يتم تنفيذ الاستعلام Subquery على كل سجل يعيده الاستعلام الرئيسي. يفيد هذا النوع من الاستعلامات الفرعية في توليد علاقة بين جدول وآخر.
وفقاً لهذه الصيغة يجب ألا يعيد الاستعلام الفرعي أكثر من قيمة واحدة لكل سجل في سجلات الاستعلام الرئيسي.

مثال:
نفرض أن لدينا الجدول Accounts الحاوي على الأرقام التسلسلية للحسابات المصرفية accountID وقيم أرصدة هذه الحسابات accountBalance. ولنفرض أن لدينا الجدول Clients الحاوي على أسماء أصحاب الحسابات clientName، ورقم الحساب لكل زبون accountID. لإظهار قائمة بأرقام الحسابات وأرصدتها وأسماء أصحابها نستخدم الصيغة:

```
Select Accounts.accountID, (select clientName from Clients where
Clients.accountID = Accounts.accountID)as myClientName,
Accounts.accountBalance
from Accounts;
```

يُعتبر المثال السابق عيّنة من الاستعلامات الفرعية المرتبطة باستعلام رئيسي نظراً لكون الاستعلام الفرعي (الظاهر باللون الأحمر) لن يعمل وحيداً، بل يحتاج إلى معلومات من الاستعلام الرئيسي وهي قيم Accounts.accountID.

انتبه:

- لاحظ أننا استخدمنا في بعض من الصيغ السابقة أسماء الجداول مع أسماء الحقول مفصولة بنقاط '.' وذلك لمنع حدوث خلط في تحديد تبعية أحد الحقول لأي من الجداول.
- لكي تعمل الصيغة في المثال السابق دون أخطاء، يُشترط أن يعيد الاستعلام الفرعي قيمة واحدة لكل سجل من سجلات الاستعلام الرئيسي.

يُعتبر استعمال الاستعلام الفرعي كعمود من أعمدة الاستعلام الرئيسي، أحد التركيبات المُستخدمة بكثرة في الاستعلامات الفرعية.

استعمال الاستعلامات الفرعية ضمن شرط Where

يأخذ هذا الاستعلام الصيغة:

```
Select columnA, columnB from Table_Name where columnB=(Subquery);
```

في هذه الصيغة أيضاً يجب أن يعيد Subquery قيمة وحيدة.

نلاحظ أن نتيجة الاستعلام الفرعي دخلت كجزء من الشرط في تعبير Where.

مثال:

لدينا الجدول Tickets الحاوي على المخالفات المرورية للعربات والجدول Owners الحاوي على أرقام لوحات العربات carNumber وأسماء أصحابها ownerName. لإظهار اسم صاحب العربة ownerName التي ارتكبت المخالفة رقم 1234، نكتب الصيغة:

```
Select ownerName, Owners.carNumber from Owners
where Owners.carNumber=(select Tickets.carNumber from tickets
where ticketNumber=1234);
```

تدخل نتيجة الاستعلام الفرعي ضمن الشرط في تعبير Where شرط أن يعيد الاستعلام الفرعي قيمة وحيدة.

الاستعلامات الفرعية التي تعيد مجموعة من القيم

كي لا يفشل الاستعلام، اشترطنا في دراستنا للاستعلامات الفرعية حتى الآن، أن يعيد الاستعلام الفرعي قيمة وحيدة.

لكي نتمكن من استخدام الاستعلامات الفرعية التي تعيد أكثر من قيمة نستخدم الصيغة التالية:

```
Select columnA, columnB from Table_Name where columnC IN(Subquery);
```

نلاحظ أننا استخدمنا هنا التعبير IN.

لكن يُشترط في الاستعلام الفرعي من هذا النوع (أي الاستعلام المُمثل بـ Subquery)، أن يعيد قيم عمود واحد فقط أي يكون من الشكل:

```
Select column1 from Table1;
```

مثال:

لنعد إلى مثال المخالفات المرورية، للحصول على قائمة بأسماء الأشخاص الذين لديهم مخالفات يكون شكل الاستعلام:

```
Select ownerName from Owners where Owners.carNumber IN
(select Distinct Tickets.carNumber from Tickets);
```

استخدمنا هنا الاستعلام الفرعي لإعادة أرقام السيارات التي لها مخالفات في سجل المخالفات مع التخلص من التكرار باستخدام Distinct.

اشترطنا حتى الآن في الاستعلامات الفرعية، سواء تلك المستخدمة كأعمدة في الاستعلام الرئيسي أو تلك المستخدمة ضمن شرط التعبير Where، أن يعيد الاستعلام الفرعي قيمة وحيدة كي لا يفشل الاستعلام.

لكي نتمكن من استخدام الاستعلامات الفرعية التي تعيد أكثر من قيمة نستخدم التعبير IN، لكن يجب أن نراعي أن يعيد الاستعلام الفرعي المستخدم مع التعبير IN حقلاً واحداً فقط (نقصد هنا حقلاً وحداً وليس قيمة واحدة).

استخدام تعبيرات Exists وAll وAny مع الاستعلامات الفرعية

التعبير Exists:

يُستخدم التعبير Exists للتحقق من إعادة الاستعلام الفرعي الذي يليه لأي سجل. ويأخذ التعبير كاملاً القيمة True في حال أرجع الاستعلام الفرعي سجلاً أو أكثر، والقيمة False إذا لم يُرجع الاستعلام الفرعي أي سجل.

يُكتب التعبير Exists وفق الصيغة:

```
Select columnA, columnB from Table_Name where Exists (Subquery);
```

مثال:

إذا كان لدينا جدول Orders بجميع الطلبيات الخاصة بشركة ، والذي يحتوي رقم الطلبية orderID ونوعها orderType ورقم الزبون clientID الذي طلبها. وجدول Clients الحاوي على أسماء الزبائن وأرقامهم clientID. لإظهار قائمة بأسماء الزبائن الذين قاموا سابقاً بإرسال طلبية من نوع "تجهيزات كهربائية"، نستخدم الصيغة:

```
Select Clients.clientName from Clients where Exists

( select * from Orders
where Orders.clientID = Clients.clientID
and
orderType='تجهيزات كهربائية');
```

استخدام تعبيرات Exists وAll وAny مع الاستعلامات الفرعية

التعبير All:

يُستخدم التعبير All للتحقق من كون جميع القيم المعادة من استعلام فرعي، تحقق شرطاً ما في تعبير Where التابع للاستعلام الرئيسي.

يُمكن أن يُكتب التعبير All وفق الصيغة:

```
Select columnA from TableA
where columnA > All from (select columnB from TableB);
```

حيث يتم هنا اختيار السجلات من الجدول A، بحيث تكون كل قيمة من قيم الحقل columnA أكبر من جميع القيم التي يعيدها الاستعلام الفرعي، أي من جميع قيم columnB في الجدول TableB.

مثال:

تريد الجهات المعنية مقارنة الأرقام Time التي سجلها العداؤون في بطولة معينة والمحفوظة مع أسمائهم Name في جدول currentRecords، بالأرقام المسجلة في جدول أرقامهم القديمة oldRecords وذلك لاستخلاص قائمة بأسماء العدائين الذين حطمت أرقامهم جميع الأرقام القديمة oldTime.

للحصول على هذه القائمة، نكتب الصيغة:

```
Select Name from currentRecords
where time < All (select oldTime from oldRecords);
```

استخدام تعبيرات Exists و All وAny مع الاستعلامات الفرعية

التعبير ANY:

يُستخدم التعبير ANY للتحقق من كون قيمة أو أكثر من القيم المعادة من استعلام فرعي، تحقق شرطاً واحداً على الأقل من شروط تعبير Where الخاص بالإستعلام الرئيسي.

يُمكن أن يُكتب التعبير Any وفق الصيغة:

```
Select columnA from TableA
where columnA > ANY from (select columnB from TableB);
```

سيتم هنا اختيار السجلات من الجدول A، حيث تكون كل قيمة من قيم الحقل columnA أكبر من قيمة واحدة على الأقل من القيم المعادة من الاستعلام الفرعي، أي من قيمة واحدة على الأقل من قيم columnB في الجدول TableB.

مثال:

تريد الجهات المعنية مقارنة الأرقام Time التي سجلها العداؤون في بطولة معينة والمحفوظة مع أسمائهم Name في جدول currentRecords، بالأرقام القياسية العالمية المسجلة في جدول bestRecords وذلك لاستخلاص قائمة بأسماء العدائين الذين حطموا أرقامهم أحد الأرقام القياسية bestTime.

للحصول على هذه القائمة نكتب الصيغة:

```
Select Name from currentRecords
where time < ANY (select bestTime from bestRecords);
```

دمج البيانات من استعلامين باستخدام التعابير Union و Intersect وExcept

تُستخدم تعابير Union وIntersect وExcept(Minus) في دمج استعلامين. وتختلف نتائج هذا الدمج بحسب التعبير المستخدم.

تُستخدم التعابير السابقة وفق الصيغة:

```
select columnA,columnB from tableA
Operator
Select columnC,columnD from tableB;
```

حيث يعبر Operator عن أحد هذه التعابير.

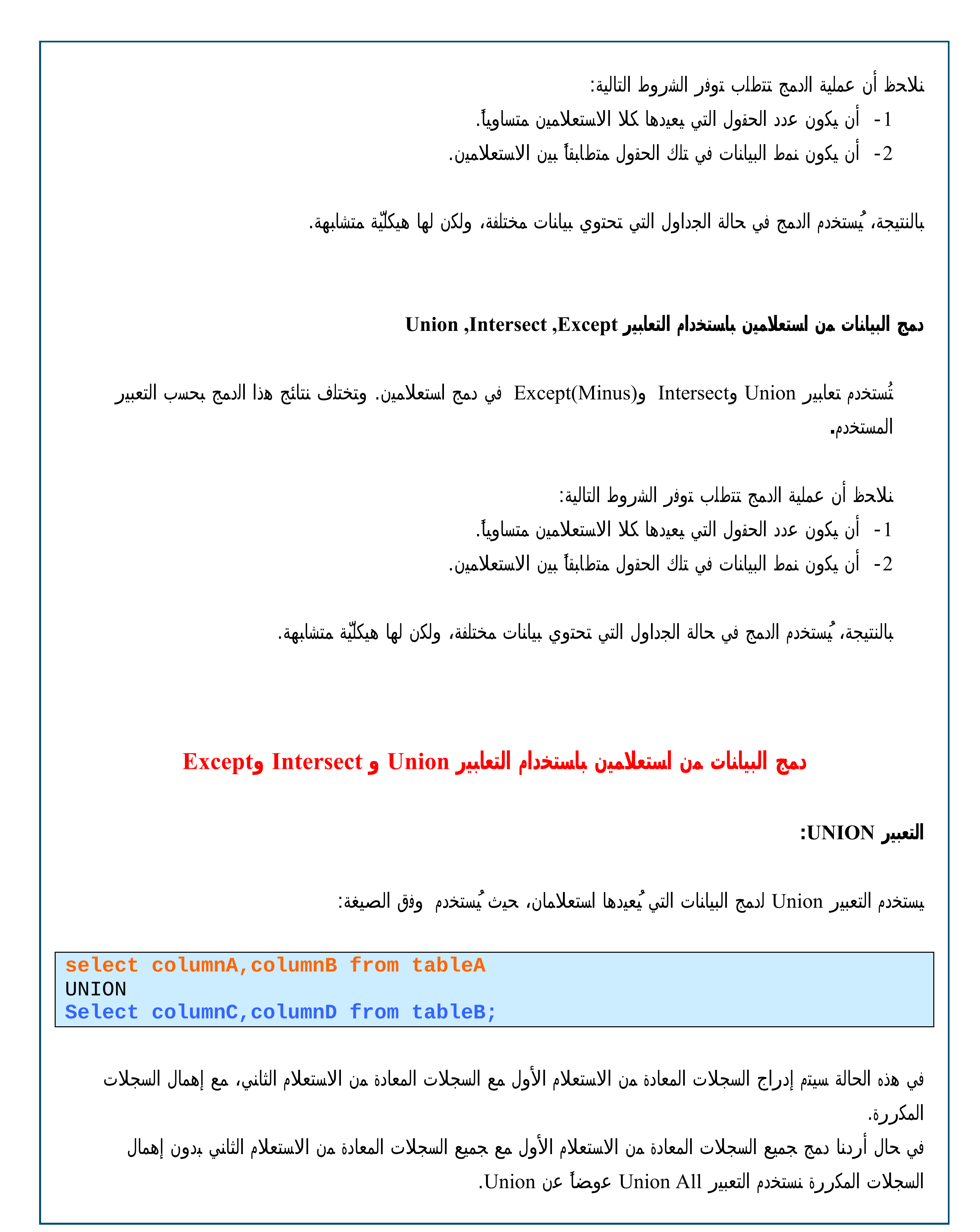
تكون هنا قائمة القيم المعادة على الشكل:

ColumnA	ColumnB
....	....
....	....
....	....
....	....
....	....

سجل من الاستعلام N1 الأول

سجل من الاستعلام N2 الثاني

حيث N1 وN2 أكبر أو تساوي الصفر.



نلاحظ أن عملية الدمج تتطلب توفر الشروط التالية:

1- أن يكون عدد الحقول التي يعيدها كلا الاستعلامين متساوياً.
2- أن يكون نمط البيانات في تلك الحقول متطابقاً بين الاستعلامين.

بالنتيجة، يُستخدم الدمج في حالة الجداول التي تحتوي بيانات مختلفة، ولكن لها هيكليّة متشابهة.

دمج البيانات من استعلامين باستخدام التعابير Union ,Intersect ,Except

تُستخدم تعابير Union وIntersect وExcept(Minus) في دمج استعلامين. وتختلف نتائج هذا الدمج بحسب التعبير المستخدم.

نلاحظ أن عملية الدمج تتطلب توفر الشروط التالية:

1- أن يكون عدد الحقول التي يعيدها كلا الاستعلامين متساوياً.
2- أن يكون نمط البيانات في تلك الحقول متطابقاً بين الاستعلامين.

بالنتيجة، يُستخدم الدمج في حالة الجداول التي تحتوي بيانات مختلفة، ولكن لها هيكليّة متشابهة.

دمج البيانات من استعلامين باستخدام التعابير Union و Intersect وExcept

التعبير UNION:

يستخدم التعبير Union لدمج البيانات التي يُعيدها استعلامان، حيث يُستخدم وفق الصيغة:

```
select columnA,columnB from tableA
UNION
Select columnC,columnD from tableB;
```

في هذه الحالة سيتم إدراج السجلات المعادة من الاستعلام الأول مع السجلات المعادة من الاستعلام الثاني، مع إهمال السجلات المكررة.

في حال أردنا دمج جميع السجلات المعادة من الاستعلام الأول مع جميع السجلات المعادة من الاستعلام الثاني بدون إهمال السجلات المكررة نستخدم التعبير Union All عوضاً عن Union.

مثال:

ليكن لدينا جدول Employees يحتوي أسماء employeeName وعلامات employeeGrade اختبار اللغة الإنكليزية للعاملين في شركة. وليكن لدينا جدول آخر Managers يحتوي أسماء managerName وعلامات managerGrade مدراء الأقسام والفروع في الشركة نفسها.

لتصدير التقرير الشامل حول امتحان اللغة، والذي تظهر فيه أسماء الناجحين في فحص اللغة، علماً أن علامة النجاح هي 50 للموظفين و60 للمدراء، نكتب الاستعلام التالي:

```
Select employeeName, employeeGrade from Employees
where employeeGrade >50
Union
Select managerName, managerGrade from Managers
where managerGrade>60;
```

دمج البيانات من استعلامين باستخدام التعابير Union ,Intersect , Except

التعبير UNION:

يُستخدم التعبير Union لدمج البيانات التي يعيدها استعلامان. في هذه الحالة يتم إدراج السجلات المعادة من الاستعلام الأول مع السجلات المعادة من الاستعلام الثاني، مع إهمال السجلات المكررة.

أما في حال أردنا دمج جميع السجلات المعادة من الاستعلام الأول مع جميع السجلات المعادة من الاستعلام الثاني بدون إهمال السجلات المكررة نستخدم التعبير Union All عوضاً عن Union.

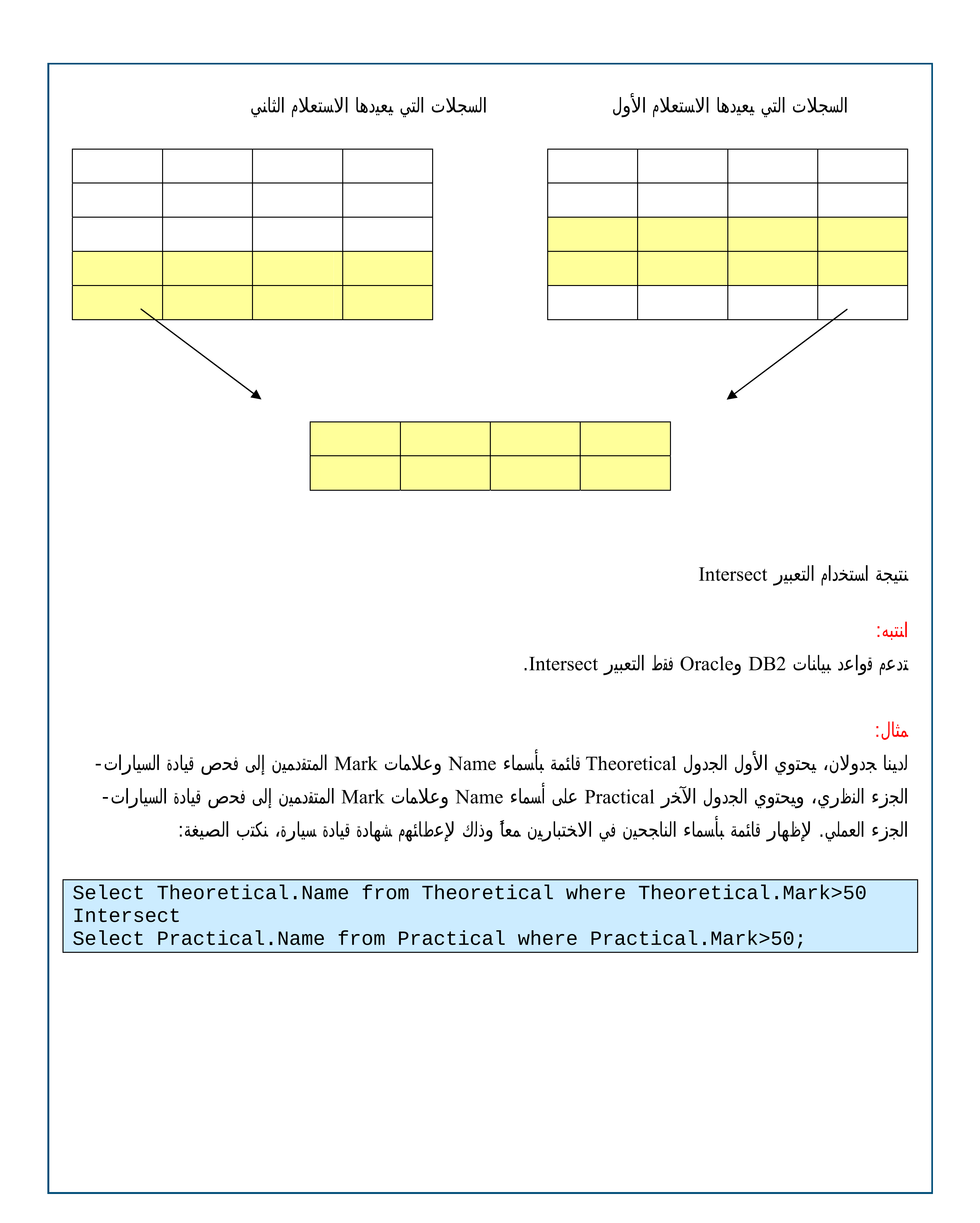
دمج البيانات من استعلامين باستخدام التعابير Union و Intersect وExcept

التعبير Intersect:

يُستخدم التعبير Intersect لاستعادة السجلات التي تظهر في كل من السجلات التي يعيدها الاستعلام الأول والسجلات التي يعيدها من الاستعلام الثاني.

يُستخدم التعبير Intersect وفق الصيغة:

```
select columnA,columnB from tableA
Intersect
Select columnC,columnD from tableB;
```



نتيجة استخدام التعبير Intersect

انتبه:

تدعم قواعد بيانات DB2 وOracle فقط التعبير Intersect.

مثال:

لدينا جدولان، يحتوي الأول الجدول Theoretical قائمة بأسماء Name وعلامات Mark المتقدمين إلى فحص قيادة السيارات-الجزء النظري، ويحتوي الجدول الآخر Practical على أسماء Name وعلامات Mark المتقدمين إلى فحص قيادة السيارات-الجزء العملي. لإظهار قائمة بأسماء الناجحين في الاختبارين معاً وذلك لإعطائهم شهادة قيادة سيارة، نكتب الصيغة:

```
Select Theoretical.Name from Theoretical where Theoretical.Mark>50
Intersect
Select Practical.Name from Practical where Practical.Mark>50;
```

دمج البيانات من استعلامين باستخدام التعابير Union و Intersect وExcept

التعبير Except:

يُستخدم التعبير Except لإعادة السجلات التي تظهر في نتيجة الاستعلام الأول ولا تظهر في نتيجة الاستعلام الثاني.

يستخدم التعبير Except وفق الصيغة:

```
select columnA,columnB from tableA
Except
Select columnC,columnD from tableB;
```

انتبه:

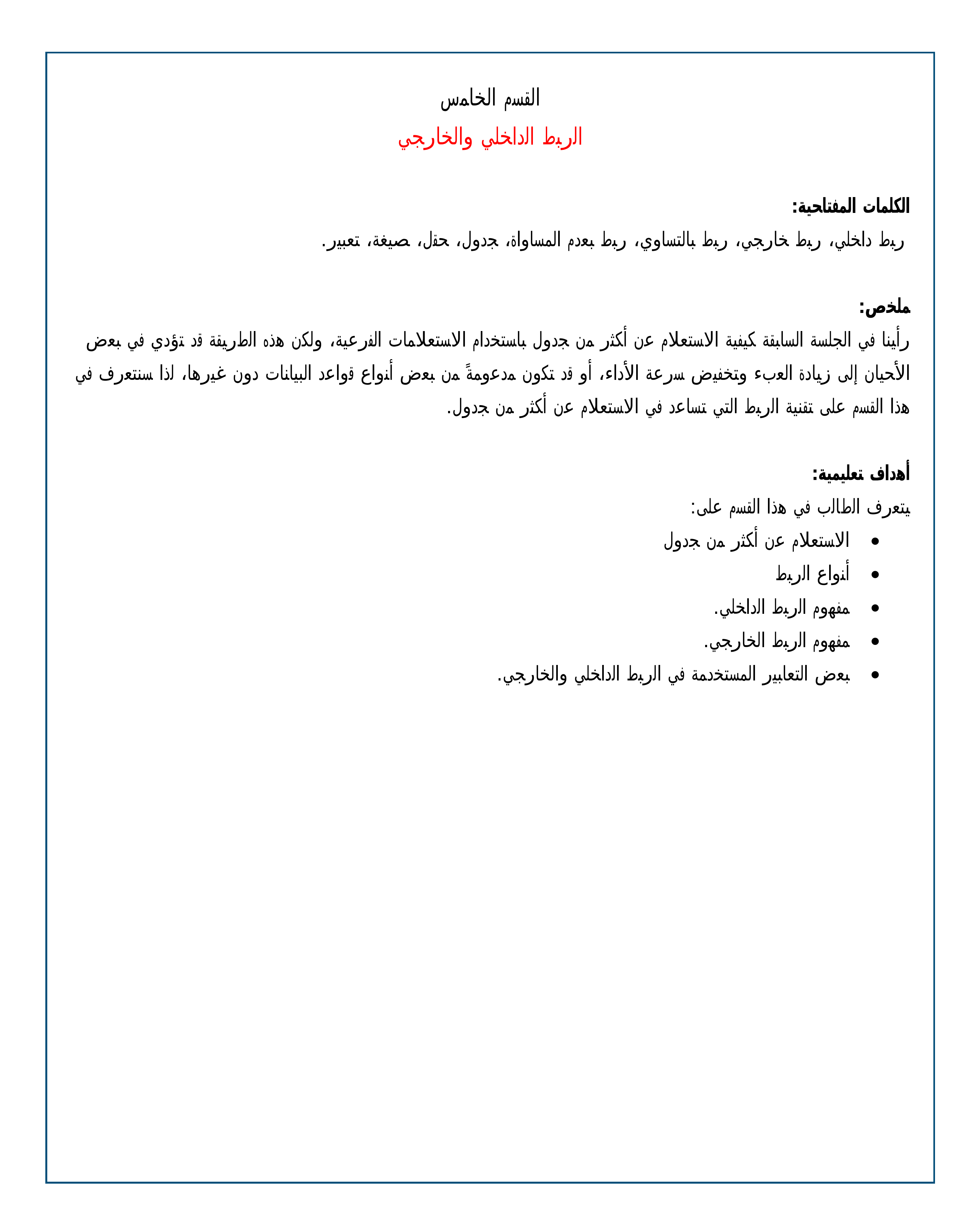
- يستعاض عن التعبير Except في Oracle بالتعبير Minus.
- التعبير Except (Minus) مدعوم فقط من قواعد بيانات DB2 وOracle.

مثال:

يعتمد مركز تأجير أفلام لحفظ معلوماته على جدولين، الأول Movies يحتوي قائمة بأسماء الأفلام movieName، ونوعها movueType، ورقمها movieNumber. ويحتوي الثاني rentMovies على قائمة بأسماء الأفلام المؤجرة حالياً مع أرقامها.

لإظهار قائمة بجميع الأفلام من النوع 'Action' غير المؤجرة نستخدم الاستعلام:

```
Select movies.movieName, Movies.movieNumber from Movies
where movieType='Action'
Minus
Select rentMovies.movieName, rentMovies.movieNumber from rentMovies;
```



القسم الخامس

الربط الداخلي والخارجي

الكلمات المفتاحية:

ربط داخلي، ربط خارجي، ربط بالتساوي، ربط بعدم المساواة، جدول، حقل، صيغة، تعبير.

ملخص:

رأينا في الجلسة السابقة كيفية الاستعلام عن أكثر من جدول باستخدام الاستعلامات الفرعية، ولكن هذه الطريقة قد تؤدي في بعض الأحيان إلى زيادة العبء وتخفيض سرعة الأداء، أو قد تكون مدعومةً من بعض أنواع قواعد البيانات دون غيرها، لذا سنتعرف في هذا القسم على تقنية الربط التي تساعد في الاستعلام عن أكثر من جدول.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- الاستعلام عن أكثر من جدول
- أنواع الربط
- مفهوم الربط الداخلي.
- مفهوم الربط الخارجي.
- بعض التعابير المستخدمة في الربط الداخلي والخارجي.



الاستعلام عن أكثر من جدول واستخدام الربط الداخلي والخارجي

سبق ورأينا في الجلسات السابقة، أن الاستعلام عن أكثر من جدول باستخدام الاستعلامات الفرعية قد وفر قدرات جيدة على معالجة البيانات. ولكن هذه القدرات قد لا تمكننا دائماً من الحصول على كل النتائج التي نحتاجها. كما أن الصيغة قد تصبح صعبة الفهم بعض الشيء وقد تؤدي إلى انخفاض مستوى الأداء أحياناً.

توفر SQL إمكانية الاستعلام عن جداول متعددة في وقت واحد باستخدام صيغة أبسط ندعوها الربط.

لا تستطيع الصيغة الجديدة استبدال كل التقنيات التي تؤمنها الاستعلامات الفرعية، ولكنها تمثل الحل الأمثل في بعض الحالات، وخاصةً في الاستعلامات التي تربط بين سجلات من جداول مختلفة.

نبدأ فيما يلي بعرض حالات الربط البسيطة لنصل بعد ذلك إلى الحالات الأكثر تعقيداً:

- الربط البسيط
- الربط بالتساوي
- الربط بعدم المساواة
- الربط الخارجي

الربط البسيط

- الربط البسيط هو استعلام عن أكثر من جدول في صيغة واحدة دون استعمال أي شرط.

- إن أبسط صيغة للتعبير عن الربط البسيط هي:

```
Select Table1.Column1, Table2.Column2 from Table1, Table2;
```

إذا فكرنا في استثمار هذه الصيغة لإعادة إسم، وصف كل طالب من الجدولين Names, Classes أول ما سيحضر إلى أذهاننا هو كتابة الصيغة:

```
Select class, Name from Classes, Names;
```

يستخدم هذا الاستعلام الربط البسيط ولكنه للأسف لن يعيد النتيجة التي نحتاجها.

- إن آلية عمل الربط البسيط في حال عدم وجود أي شرط مرافق هي عبارة عن عملية جداء ديكارتي لقيم الحقول المحددة من الجدولين. فإذا كانت قيم الحقل المطلوب من الجدول الأول هي {A, B, C} وقيم الحقل المطلوب من الجدول الثاني هي{D, E, F} فسيكون عدد القيم المعادة 9 قيم هي التالية:

{(A,D), (A,E), (A,F), (B,D), (B,E), (B,F), (C,D), (C,E), (C,F)}

بالنتيجة، إذا كان لدينا 100 سجل في كل من الجدولين المربوطين ربطاً بسيطاً سنحصل على 10000 سجل يعيدها استعلام الربط البسيط.

- يسمى استعلام الربط البسيط أيضاً بالربط المتصالب. ويمكن التعبير عن نفس صيغة الربط السابقة، بالصيغة:

```
Select Table1.Column1, Table2.Column2 from Table1 Cross Join Table2;
```

مثال:

يريد كيميائي اختبار تأثير مجموعة من المواد الكيميائية على مجموعة من المنتجات التي تصنعها الشركة التي يعمل لديها. فإذا كانت أسماء المنتجات (productName) مدرجة في جدول Products وأسماء المواد المراد اختبار تأثيرها (materialName) مدرجة في جدول ChemicalEffects، المطلوب مساعدة هذا الكيميائي في تجهيز لائحة الاختبارات.

الحل:

```
Select productName, materialName from Products, ChemicalEffects;
```

أو

```
Select productName, materialName from Products Cross Join
ChemicalEffects;
```

انتبه:

لا تدعم قواعد بيانات DB2 التعبير Cross Join

الربط البسيط

الربط البسيط هو استعلام عن أكثر من جدول في صيغة واحدة دون استعمال أي شرط.

إن آلية عمل الربط البسيط في حال عدم وجود أي شرط مرافق هي عبارة عن عملية جداء ديكارتي لقيم الحقول المحددة من الجداول المرتبطة. فإذا كانت قيم الحقل المطلوب من الجدول الأول هي {A, B, C} وقيم الحقل المطلوب من الجدول الثاني هي {D, E, F} فسيكون عدد القيم المعادة 9 قيم وهي التالية:

{(A,D), (A,E), (A,F), (B,D), (B,E), (B,F), (C,D), (C,E), (C,F)}

بالنتيجة، إذا كان لدينا 100 سجل في كل من الجدولين المربوطين ربطاً بسيطاً سنحصل على 10000 سجل يعيدها استعلام الربط البسيط.

يسمى استعلام الربط البسيط أيضاً بالربط المتصالب.

الربط بالتساوي

- يُعرّف الربط بالتساوي على أنه الربط البسيط بين سجلات جدول أول، وسجلات جدول ثان اعتماداً على مساواة بين قيمة حقل في سجل من الجدول الأول وقيمة حقل في سجل من الجدول الثاني.

- يُعبّر عن الربط بالتساوي بالصيغة:

```
Select Table1.Column1, Table1.Column2, Table2.Column3
From Table1, Table2 where Table1.Column1 = Table2.Column2;
```

لاحظ أننا قد استخدمنا مساواة بين الحقل الأول من الجدول الأول Table1.Column1 والحقل الثاني من الجدول الثاني Table2.Column2 مما يعني أن عملية الربط لاتستخدم بالضرورة نفس الحقول التي يجب أن يعيدها الاستعلام، أي: Table1.Column1, Table1.Column2, Table2.Column3

- يمكن الوصول إلى نفس النتيجة السابقة باستخدام الصيغة:

```
Select Table1.Column1, Table1.Column2, Table2.Column3
From Table1
Join Table2
ON Table1.Column1 = Table2.Column2;
```

مثال:

إذا كان لدينا جدول Names يحوي أسماء جميع الأشخاص (name) المقيمين في قرية مع أرقامهم التأمينية(INumber) والجدول Addresses الذي يتضمن عناوين المنازل (address) في تلك القرية مع الرقم التأميني (INumber) للمقيم في المنزل، وأردنا كتابة الاستعلام الذي يعيد قائمة بأسماء الأشخاص وعناوينهم، يمكننا استخدام الصيغة:

```
Select Names.name, Addresses.address from Names, Addresses
Where Names.INumber = Addresses.INumber;
```

أو

```
Select Names.name, Addresses.address
From Names
Join Addresses
ON Names.INumber = Addresses.INumber;
```

استخدام الربط بالتساوي مع عدة جداول

لا تتوقف إمكانيات الربط بالتساوي عند حدود الربط بين جدولين فقط بل يمكن أن تتعداها إلى مجموعة من الجداول.
نستخدم لإتمام هذه العملية الصيغة:

```
Select Table1.Column1, Table2.Column2, Table3.Column4
From Table1 Join Table2
ON Table1.Column1 = Table2.Column2
Join Table3
ON Table1.Column3 = Table3.Column4;
```

هنا يتم الربط بين الجداول اعتماداً على أسماء الحقول المتساوية المحددة بعد التعبير ON. ففي حالة الصيغة السابقة ربطنا الجدول Table1 مع Table2 معتمدين على تساوي قيم العمود Column1 من الجدول Table1 مع قيم العمود Column2 من الجدول Table2، وربطنا الجدول Table3 بالجدول Table1 معتمدين على تساوي قيم العمود Column3 من الجدول Table1 مع قيم العمود Column4 من الجدول Table3.

مثال:
لدينا الجدول Customers الحاوي على رمز الزبون customerID، اسمه customerName والجدول CreditCards الحاوي على رقم بطاقة الزبون الائتمانية cardNumber وعلى رقمه customerID، والجدول Addresses الحاوي على عناوين الزبائن وكان المطلوب إعادة قائمة برقم الزبون وأسمه، ورقم بطاقته الإئتمانية، والبلد الذي يقيم فيه country. للحصول على المطلوب، نكتب الاستعلام:

```
Select Customers.customerID, Customers.customerName,
CreditCards.cardNumber, Addresses.country
From Customers
Join CreditCards
ON Customers.customerID = CreditCards.customerID
Join Addresses
ON Customers.customerID = Addresses.customerID;
```

استخدام الربط بالتساوي مع عدة جداول (تتمة)

قد لا تعمل الصيغة السابقة على نظام إدارة قواعد البيانات MS Access لذلك قد يفضل البعض استخدام تقنية الربط المتساوي المتداخل أي استخدام الصيغة:

```
Select Table1.Column1, Table2.Column2, Table3.Column4
From Table2
Inner Join
(Table3 Inner Join Table1 ON Table3.Column4 = Table1.Column1)
ON Table2.Column2 = Table1.Column1;
```

تشبه هذه الصيغة تلك التي استخدمناها في الشريحة السابقة ولكن MS Access يواجه صعوبة في ترجمة عمليات الربط بصورة مباشرة، لذلك لجأنا إلى توليد بنى ربط متداخلة مجمعة بأقواس. فكما نلاحظ قمنا بربط الجدول Table3 مع الجدول Table1 بتساوي الحقلين Column4 من الجدول Table3 مع الحقل Column1 من الجدول Table1، ثم تعاملنا مع التعبير كاملاً كطرف في عملية الربط مع الجدول Table2 بتساوي الحقل Column2 من الجدول Table2 مع الحقل Column1 من الجدول Table1.

مثال:

إذا كان لدينا مخزن ألبسة يعتمد قاعدة بيانات تحتوي ثلاثة جداول: الأول جدول الأقسام Sectors (ولادي- نسائي- سن محير...)، الذي يحتوي على رمز القسم sectorID وإسم القسم sectorName. والثاني جدول الفصول Seasons الذي يحتوي رمز الفصول والسنوات (خريف 2004 – شتاء2003...) seasonID، وأسمائها seasonInfo. والثالث جدول المنتجات Products الذي يحتوي الأرقام التسلسلية للقطع productID، وتوصيفها productDescription، وأسعارها productPrice، ورمز القسم seasonID، ورمز القسم sectorID الذي تتبع له.

والمطلوب إظهار قائمة بتوصيف القطع وأسعارها، والأقسام التي تتبع لها والقسم الخاص بها.

الحل:

```
Select productDescription, productPrice, seasonName, sectorName
From Sectors
Inner Join
(Seasons Inner Join Products ON Seasons.seasonID = Products.seasonID)
ON Sectors.sectorID = Products.sectorID;
```

قد لا تعمل الصيغة السابقة على نظام إدارة قواعد البيانات MS Access لذلك قد يفضل البعض استخدام تقنية الربط المتساوي المتداخل. إذ يواجه MS Access صعوبة في ترجمة عمليات الربط بصورة مباشرة، لذلك نلجىء عادةً إلى توليد بنى ربط متداخلة مجمعة بأقواس.

الربط باللامساواة

يعتمد الربط بالمساواة على استخدام المساواة في شرط التعبير Where ولكن هذا لا يعني أننا لا نستطيع استخدام عمليات المقارنة الأخرى (أكبر، أصغر، وغيرها) كما في الصيغة التالية:

```
Select Table1.Column1, Table2.Column2
From Table1, Table2
Where Table1.Column1 < Table2.Column2;
```

مثال:

لدينا جدول Stores الخاص بمعمل أقمشة، والذي يحتوي يحتوي أسماء المخازن storeName وأرقامها storeID وجدول آخر Occupation يحتوي معلومات المشغولية للمخازن التي تتضمن رقم المخزن stroreID وكمية البضاعة في المخزن quantity ونوعها Type.

لإعادة قائمة بالمخازن الفارغة غير المشغولة نكتب الاستعلام:

```
Select  Stores.storeID, Stores.storeName from Stores, Occupation
Where Stores.storeID <> Occupation.storeID
```

الربط الخارجي

لنتمكن من فهم الربط الخارجي يجب أن نعود إلى فكرة الربط بالتساوي حيث استخدمنا التعبير Inner Join.

في حالة Inner Join، كانت السجلات التي أرجعها الاستعلام، هي السجلات التي تحقق شرط الربط الذي يظهر بعد تعبير ON، حيث تم اسقاط السجلات غير المتطابقة من جدول النتائج. أما في حالة الربط الخارجي Outer Join فلا يتم إسقاط السجلات غير المتطابقة.

للربط الخارجي ثلاثة أنواع: Left, Right, Full.

- لأخذ جميع السجلات من الجدول الأول Table1 وفقط السجلات من الجدول الثاني Table2 التي تتطابق فيها قيمة الحقل Column1 من الجدول Table1 مع قيمة الحقل Column2 من الجدول الثاني Table2، نكتب الصيغة:

```
Select * from Table1 LEFT OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

- لأخذ جميع السجلات من الجدول الثانيTable2 وفقط السجلات من الجدول الأول Table1 التي تتطابق فيها قيمة الحقل Column1 من الجدول Table1 مع قيمة الحقل Column2 من الجدول الثاني Table2، نكتب الصيغة:

```
Select * from Table1 RIGHT OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

- لأخذ جميع السجلات من الجدول الثانيTable2 وجميع السجلات من الجدول الأول Table1 بحيث تتوضع السجلات التي تتطابق فيها قيمة الحقل Column1 من الجدول Table1 مع قيمة الحقل Column2 من الجدول الثاني Table2 في نفس السجل من جدول القيم المعادة، نكتب الصيغة:

```
Select * from Table1 FULL OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

ينتج عن عمليات الربط الخارجي، في الحالة العامة، سجلات تحتوي في حقول معينة القيمة NULL بسبب اختلاف عدد السجلات التي نريد ربطها، وهذا ما سنوضحه بالتفصيل لاحقاً مع مثال مناسب لكل نوع من أنواع الربط الخارجي.

الربط الخارجي

لنتمكن من فهم الربط الخارجي يجب أن نعود إلى فكرة الربط بالتساوي حيث استخدمنا التعبير Inner Join.

في حالة Inner Join، كانت السجلات التي أرجعها الاستعلام، هي السجلات التي تحقق شرط الربط الذي يظهر بعد تعبير ON، حيث تم اسقاط السجلات غير المتطابقة من جدول النتائج. أما في حالة الربط الخارجي Outer Join فلا يتم إسقاط السجلات غير المتطابقة.

للربط الخارجي ثلاثة أنواع: Left وRight وFull. ينتج عن عمليات الربط الخارجي، في الحالة العامة، سجلات تحتوي في حقول

معينة القيمة NULL بسبب اختلاف عدد السجلات التي نريد ربطها، وهذا ما سنوضحه بالتفصيل لاحقاً مع مثال مناسب لكل نوع من أنواع الربط الخارجي.

Left Join

لنستخدم صيغة الربط الخارجي التالية على الحقلين Column1 من الجدول الأول Table1، وColumn2 من الجدول الثاني Table2:

```
Select * from Table1 LEFT OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

يُرجع هذا النوع من أنواع الربط الخارجي:

- جميع القيم الخاصة بالحقل التابع للجدول الأول،
- جميع القيم المطابقة لها والخاصة بالحقل التابع للجدول الثاني، حيث يتم إدراج القيمة Null في قيم الحقل التابع للجدول الثاني وذلك في السجلات التي لا تكون فيها قيمة حقل الجدول الأول مطابقة لقيمة حقل الجدول الثاني.

كما يكون عدد السجلات المعادة مطابقاً لعدد السجلات التي يعيدها الاستعلام عن قيم حقل الجدول الأول.

مثال:

لنستخدم الصيغة:

```
Select * from Table1 LEFT OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

إذا كان لدينا جدول Table1 يحتوي في الحقل Column1 القيم {1, 5, 8, 3} والجدول Table2 الذي يحتوي في الحقل Column2 القيم {6, 5, 7, 9} فإن تطبيقها سيولد النتائج التالية:

Column1	Column2
1	Null
5	5
8	Null
3	Null

نلاحظ أنه تم إدراج القيمة Null لقيم الحقل Column2 في السجلات التي لم تكن فيها قيمة الحقل Column2 مطابقة لقيمة الحقل Column1.

كما يكون عدد السجلات المعادة مطابقاً لعدد السجلات التي يعيدها الاستعلام عن قيم الحقل Column1 من الجدول Table1.

عندما نطبق هذا النوع من أنواع الربط الخارجي على حقلين من جدولين مختلفين، نحصل على:

- جميع القيم الخاصة بالحقل التابع للجدول الأول،
- جميع القيم المطابقة لها والخاصة بالحقل التابع للجدول الثاني، حيث يتم إدراج القيمة Null في قيم الحقل التابع للجدول الثاني وذلك في السجلات التي لا تكون فيها قيمة حقل الجدول الأول مطابقة لقيمة حقل الجدول الثاني.

كما يكون عدد السجلات المعادة مطابقاً لعدد السجلات التي يعيدها الاستعلام عن قيم حقل الجدول الأول.

Right Join

لنستخدم صيغة الربط الخارجي التالية على الحقلين Column1 من الجدول الأول Table1، وColumn2 من الجدول الثاني Table2:

```
Select * from Table1 RIGHT OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

يُرجع هذا النوع من أنواع الربط الخارجي:

- جميع القيم الخاصة بالحقل التابع للجدول الثاني،
- جميع القيم المطابقة لها والخاصة بالحقل التابع للجدول الأول، حيث يتم إدراج القيمة Null في قيم الحقل التابع للجدول الأول وذلك في السجلات التي لا تكون فيها قيمة حقل الجدول الثاني مطابقة لقيمة حقل الجدول الأول.

كما يكون عدد السجلات المعادة مطابقاً لعدد السجلات التي يعيدها الاستعلام عن قيم حقل الجدول الثاني.

مثال:

لنستخدم الصيغة:

```
Select * from Table1 RIGHT OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

إذا كان لدينا جدول Table1 يحتوي في الحقل Column1 القيم {1, 5, 8, 3} والجدول Table2 الذي يحتوي في الحقل Column2 القيم {6, 5, 7, 9} فإن تطبيقها سيولد النتائج التالية:

Column1	Column2
Null	6
5	5
Null	7
Null	9

نلاحظ أنه تم إدراج القيمة Null لقيم الحقل Column2 في السجلات التي لم تكن فيها قيمة الحقل Column2 مطابقة لقيمة الحقل Column1.

كما يكون عدد السجلات المعادة مطابقاً لعدد السجلات التي يعيدها الاستعلام عن قيم الحقل Column2 من الجدول Table2.

عندما نطبق هذا النوع من أنواع الربط الخارجي على حقلين من جدولين مختلفين، نحصل على:

- جميع القيم الخاصة بالحقل التابع للجدول الثاني،
- جميع القيم المطابقة لها والخاصة بالحقل التابع للجدول الأول، حيث يتم إدراج القيمة Null في قيم الحقل التابع للجدول الأول وذلك في السجلات التي لا تكون فيها قيمة حقل الجدول الثاني مطابقة لقيمة حقل الجدول الأول.

كما يكون عدد السجلات المعادة مطابقاً لعدد السجلات التي يعيدها الاستعلام عن قيم حقل الجدول الثاني.

Full Join

لنستخدم صيغة الربط الخارجي التالية على الحقلين Column1 من الجدول الأول Table1، وColumn2 من الجدول الثاني Table2:

```
Select * from Table1 FULL OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

يُرجع هذا النوع من أنواع الربط الخارجي:

- جميع القيم الخاصة بالحقل التابع للجدول الأول، حيث يتم إدراج القيمة Null في قيم الحقل التابع للجدول الأول وذلك في السجلات التي لا تكون فيها قيمة حقل الجدول الثاني مطابقة لقيمة حقل الجدول الأول.
- جميع القيم المطابقة لها والخاصة بالحقل التابع للجدول الثاني، حيث يتم إدراج القيمة Null في قيم الحقل التابع للجدول الثاني وذلك في السجلات التي لا تكون فيها قيمة حقل الجدول الأول مطابقة لقيمة حقل الجدول الثاني.

مثال:

لنستخدم الصيغة:

```
Select * from Table1 FULL OUTER JOIN Table2
ON Table1.Column1 = Table2.Column2;
```

إذا كان لدينا جدول Table1 يحتوي في الحقل Column1 القيم {1, 5, 8, 3} والجدول Table2 الذي يحتوي في الحقل Column2 القيم {6, 5, 7, 9} فإن تطبيقها سيولد النتائج التالية:

Column1	Column2
1	Null
5	5
8	Null
3	Null
Null	6
Null	7
Null	9

نلاحظ أنه تم إدراج القيمة Null لقيم الحقل Column1 في السجلات التي لم تكن فيها قيمة الحقل Column1 مطابقة لقيمة الحقل Column2 وإدراج القيمة Null لقيم الحقل Column2 التي لم تتطابق فيها قيمة Column1 مع Column2.

Full Join

عندما نطبق هذا النوع من أنواع الربط الخارجي على حقلين من جدولين مختلفين، نحصل على:

- جميع القيم الخاصة بالحقل التابع للجدول الأول، حيث يتم إدراج القيمة Null في قيم الحقل التابع للجدول الأول وذلك في السجلات التي لا تكون فيها قيمة حقل الجدول الثاني مطابقة لقيمة حقل الجدول الأول.
- جميع القيم المطابقة لها والخاصة بالحقل التابع للجدول الثاني، حيث يتم إدراج القيمة Null في قيم الحقل التابع للجدول الثاني وذلك في السجلات التي لا تكون فيها قيمة حقل الجدول الأول مطابقة لقيمة حقل الجدول الثاني.

استخدام Natural Join

يقوم التعبير Natural Join بعملية ربط اعتماداً على الحقول ذات الأسماء المُشتركة بين الجدولين والتي تحتوي على قيم متطابقة. يأخذ التعبير الصيغة:

```
Select Table1.Column1, Table2.Column1 from Table1 Natural Join
Table2;
```

نلاحظ هنا أننا لم نستخدم التعبير ON وشرط التساوي لأنهما مُستخدمان ضمنياً في التعبيرNatural Join حيث يتم تحقيق الربط عن طريق استخدام الحقل الذي يشترك الجدولان بإسمه.

مثال:

ليكن لدينا الجدول Pictures الحاوي على الأرقام التسلسلية للصور (pictureID) وعام تصويرها بالإضافة إلى شرح عن الصورة (pictureDescription). وليكن لدينا الجدول Names الحاوي على أسماء أصحاب الصور (clientName) والأرقام التسلسلية للصور (pictureID). فإذا كان المطلوب إعادة قائمة باسم كل شخص ووصف الصورة الخاصة به يمكننا كتابة الصيغة:

```
Select clientName, pictureDescription from Names Natural Join
Pictures;
```

نلاحظ في الصيغة السابقة أن الربط قد تم اعتماداً على الحقل الذي يشترك الجدولان بإسمه وهو pictureID.

استخدام Natural Join

يقوم التعبير Natural Join بعملية ربط لجميع الحقول ذات الأسماء المُشتركة والتي تحتوي على قيم متطابقة من الجدولين.
في هذا الاستعلام لا نستخدم التعبير ON وشرط التساوي لأنهما مُستخدمان ضمنياً في التعبير Natural Join حيث يتم تحقيق الربط عن طريق استخدام الحقل الذي يشترك الجدولان بإسمه.

الربط باستخدام التعبير Using

يستخدم التعبير Using في حال كان اسم الحقل نعتمد على قيمه في عملية الربط متشابهاً في الجدولين.

يأخذ التعبير Using الصيغة:

```
Select Table1.Column2, Table2.Column3
From Table1 Join Table2 Using (Column1)
```

اعتبرنا هنا أن الربط يتم اعتماداً على قيم الحقل Column1 من الجدول الأول وColumn1 من الجدول الثاني.

انتبه:

إن التعبيرين Using وNatural Joing مدعومان من قواعد بيانات Oracle في نسختها 9I

يستخدم التعبير Using في حال كان اسم الحقل نعتمد على قيمه في عملية الربط متشابهاً في الجدولين.

تمرين عام

حل التمرين التالي:

ليكن لدينا الجداول Customers وOrders وCoutries .

- يحتوي الجدول Customers على معلومات الزبون (customerInfo) إضافة إلى رمز البلد (countryID) الذي ينتمي إليه الزبون.
- ويحتوي الجدول Orders على رقم الزبون (customerID) وعلى الطلبات (orderDescription) التي قام بعض الزبائن بطلبها.
- ويحتوي الجدول Orders على رقم البلد (countryID)، وأسمه (countryName) ومعلومات عنه (countryInfo).

المطلوب إظهار قائمة بجميع الطلبات مع أسماء الزبائن الذين قاموا بطلبها، إن توفرت أسماؤهم، مع إظهار اسم البلد الذي ينتمي إليه الزبون.

الحل:

```
Select orderDescription, customerName, countryName
From Countries Inner Join (customers RIGHT OUTER JOIN Orders
ON Customers.customerID = Orders.customerID
ON Countries.countryID = Customers.countryID;
```

نلاحظ أننا استخدمنا تداخل ربط خارجي بين الجدول Customers والجدول Orders وربط داخلي بين الجدول Countries والجدول Customers.

حل التمرين التالي:

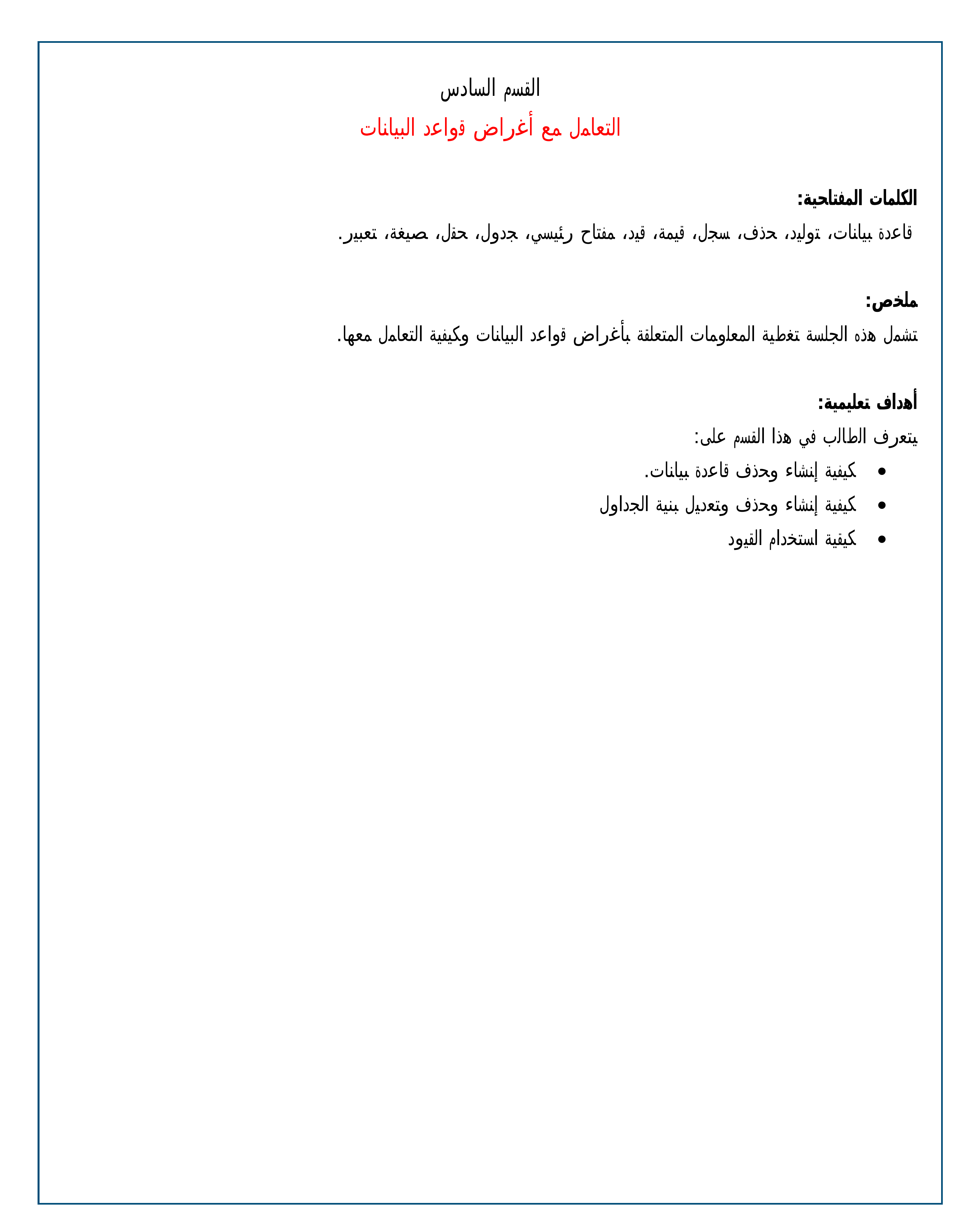
ليكن لدينا الجداول Customers وOrders وCoutries .

- يحتوي الجدول Customers على معلومات الزبون (customerInfo) إضافة إلى رمز البلد (countryID) الذي ينتمي إليه الزبون.
- ويحتوي الجدول Orders على رقم الزبون (customerID) وعلى الطلبات (orderDescription) التي قام بعض الزبائن بطلبها.
- ويحتوي الجدول Orders على رقم البلد (countryID)، وأسمه (countryName) ومعلومات عنه (countryInfo).

المطلوب إظهار قائمة بجميع الطلبات مع أسماء الزبائن الذين قاموا بطلبها، إن توفرت أسماؤهم، مع إظهار اسم البلد الذي ينتمي إليه الزبون.

الحل:

نستخدم تداخل ربط خارجي بين الجدول Customers والجدول Orders وربط داخلي بين الجدول Countries والجدول Customers، كما هو موضح في صيغة الحل.



القسم السادس

التعامل مع أغراض قواعد البيانات

الكلمات المفتاحية:

قاعدة بيانات، توليد، حذف، سجل، قيمة، قيد، مفتاح رئيسي، جدول، حقل، صيغة، تعبير.

ملخص:

تشمل هذه الجلسة تغطية المعلومات المتعلقة بأغراض قواعد البيانات وكيفية التعامل معها.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- كيفية إنشاء وحذف قاعدة بيانات.
- كيفية إنشاء وحذف وتعديل بنية الجداول
- كيفية استخدام القيود

التعامل مع أغراض قواعد البيانات

ركّزنا حتى الآن، وفي كل المواضيع التي تطرقنا إليها، على الاستعلام عن جداول في قاعدة بيانات معتبرين أن تلك قواعد البيانات منشأة مسبقاً والجداول موجودة. إلا أننا لم نتطرق إلى كيفية إنشاء قواعد البيانات: الجداول والفهارس والقيود عليها. وهذا ما يندرج تحت عنوان التعامل مع أغراض قواعد البيانات.

سنتعرف من خلال هذه الجلسة والجلسة القادمة على:

- كيفية إنشاء وحذف قاعدة بيانات.
- كيفية إنشاء وحذف وتعديل بنية الجداول.
- كيفية إنشاء وحذف وتعديل القيود على الحقول.
- كيفية إنشاء وحذف وتعديل الجداول المؤقتة وكيفية التعامل معها.
- كيفية إنشاء وحذف وتعديل الفهارس.
- كيفية إنشاء وحذف وتعديل المعاينات.

توليد وحذف قاعدة بيانات

تختلف طريقة تخزين وإدارة البيانات بين أنظمة إدارة قواعد البيانات. إلا أننا سنركز على التقنيات الأكثر شيوعاً.

* توليد قاعدة بيانات:

في أنظمة إدارة قواعد البيانات Oracle وSQL Server وMySQL وDB2 يمكنك ببساطة استخدام الصيغة:

```
CREATE DATABASE database_name;
```

* حذف قاعدة بيانات:

يمكننا حذف قاعدة بيانات ما باستخدام الصيغة:

```
DROP DATABASE database_name;
```

إلا أن التعبير (DROP DATABASE) مدعوم فقط من قواعد بيانات SQL Server وMy SQL وDB2، أما في Oracle فيمكننا استخدام نفس التوليد (CREATE DATABASE) لحذف قاعدة بيانات، كما يمكننا أن نستخدم تطبيق Oracle Database Assistant لحذف قاعدة بيانات.

انتبه:

- إن استخدام (CREATE DATABASE) مع إسم قاعدة بيانات موجودة في Oracle يؤدي إلى إلغاء واستبدال تلك القاعدة فتوخ الحذر عند استخدام هذا التعبير.
- تمتلك أنظمة إدارة قواعد البيانات آليات خاصة لتوليد قاعدة بيانات دون الحاجة إلى كتابة الصيغة الآنفة الذكر. فمثلاً يستخدم SQL Server تطبيق Enterprise Manager لتنفيذ عملية التوليد، وتستخدم قواعد بيانات DB2 تطبيق Control Center لنفس الغرض.
- لايدعم MS Access أياً من تعابير إنشاء وحذف قواعد البيانات لذلك نلجأ لاستخدام الخيار New من القائمة File في نافذة التطبيق Access. ولحذف قاعدة بيانات Access يكفي حذف ملف قاعدة البيانات تلك والذي يحمل اللاحقة (.mdb).

تختلف طريقة تخزين وإدارة البيانات بين أنظمة إدارة قواعد البيانات. إلا أننا سنركز على التقنيات الأكثر شيوعاً.

لتوليد قاعدة بيانات في أنظمة إدارة قواعد البيانات Oracle وSQL Server وMySQL وDB2 يمكنك ببساطة استخدام التعبير (CREATE DATABASE)

أما حذف قاعدة بيانات فيتم باستخدام التعبير (DROP DATABASE)

إلا أن التعبير (DROP DATABASE) مدعوم فقط من قواعد بيانات SQL Server وMy SQL وDB2، أما في Oracle فيمكننا استخدام نفس التوليد (CREATE DATABASE) لحذف قاعدة بيانات، كما يمكننا أن نستخدم تطبيق Oracle Database Assistant لحذف قاعدة بيانات.

إنشاء وحذف الجداول

لإنشاء جدول نستخدم الصيغة:

```
CREATE TABLE table_name
(column1_name column1_data_type column1_constraints,
column2_name column2_data_type column2_constraints,…);
```

لحذف جدول نستخدم الصيغة:

```
DROP TABLE table_name;
```

لتفريغ جميع السجلات من جدول ما نستخدم الصيغة:

```
TRUNCATE TABLE table_name;
```

مثال:

إذا أردنا إنشاء قاعدة بيانات Store وأردنا إنشاء جدولين ضمنها أحدهما باسم Customers والآخر باسم Products.
الجدول Customers يحوي الرقم التسلسلي للزبون (ID) واسم الزبون (name) ورقم هاتفه (phone). والجدول Products يحتوي الرقم التسلسلي (ID) لمنتج ووصف له (description).
لإنشاء هذه البنية نكتب الصيغة:

```
CREATE DATABASE Store;

CREATE TABLE Customers
(ID Int, name varchar(50), phone varchar(15);

CREATE TABLE Products
(ID Int, description varchar(75));
```

يمكننا بعد إنشاء الجداول البدء بإدراج سجلات ضمنها بالصيغة التي تعرفنا عليها في جلسات سابقة:

```
Insert into products (ID,description) values (1,'HPComputer');
```

نلاحظ هنا استخدامنا لأنواع البيانات Int وvarchar.
لإظهار جدول تفصيلي بأهم أنواع البيانات إضغط على الوصلة: جدول بأهم أنواع البيانات.

انتبه:

- لا يمكن عَكس عملية حذف جدول من قاعدة البيانات في MySQL أي لا يمكن استرجاع الجدول بعد حذفه بعكس SQL Server وOracle وDB2 حيث يمكن التراجع عن الحذف واستعادة البيانات في حال تم إجراء تلك العملية كجزء من مناقلة.
- يمكننا في SQL Server, Oracle, MySQL إخلاء الجدول عوضاً عن حذفه أي تفريغ جميع السجلات، وهي أيضاً عملية غير عكوسة ولكنها أسرع من استخدام الصيغة:

```
DELETE from table_name
```

إنشاء وحذف الجداول

تستخدم التعابير (CREATE TABLE) و(DROP TABLE) و(TRUNCATE TABLE) لتوليد وحذف وإفراغ الجداول. وتوضح الصيغ الظاهرة كيفية استخدام كل منها.

نسخ الجداول

لنسخ جدول ما أثناء توليده يمكننا استخدام الصيغة:

```
CREATE TABLE table_name_copy AS Select* from table_name;
```

تتغير الصيغة بشكل طفيف في SQL Server وتصبح على الشكل:

```
Select * Into table_name_copy from table_name;
```

وفي MySQL تصبح الصيغة:

```
CREATE TABLE table_name_copy Select* from table_name;
```

في جميع الصيغ السابقة تمثل table_name إسم الجدول اذي نود نسخ بنيته مع البيانات الموجودة فيه وtable_name_copy الجدول المنسوخ عن الجدول table_name.

في حال أردنا نسخ بنية الجدول فقط بدون نسخ السجلات في الجدول يمكننا وضع شرط في تعبير Where له نتيجة False دائماً أي تصبح الصيغة على الشكل:

```
CREATE TABLE table_name_copy AS Select* from table_name
Where 1 = 0;
```

نلاحظ أن الشرط 1 = 0 لن يتحقق أبداً لذا سيتم نسخ بنية الجدول فقط ولن يوجد أي سجل سيحقق الشرط 1= 0.

أما في قواعد بيانات DB2 فسنضطر إلى إضافة التعبير DEFINITION ONLY فتصبح الصيغة من الشكل:

```
CREATE TABLE table_name_copy AS (Select* from table_name) DEFINITION
ONLY;
```

مثال:

لنسخ الجدول Logs إلى جدول آخر يسمى OldLogs نستخدم الصيغة:

```
CREATE TABLE OldLogs AS Select * from Logs;
```

لنسخ جدول ما أثناء توليده يمكننا استخدام تعبير (CREATE TABLE) أيضاً مع تغييرات طفيفة على صيغة الإستعلام بين Oracle وMySQL وSQLServer.

أما في حال أردنا نسخ بنية الجدول فقط بدون نسخ السجلات في الجدول فيمكننا وضع شرط في تعبير Where له نتيجة False دائماً، كوضع 0=1 على سبيل المثال.

وتشذ قواعد بيانات DB2 على ماسبق وتضطرنا لإضافة التعبير DEFINITION ONLY من أجل سخ بنية الجدول فقط دون سجلاتها.

تعديل بنية الجداول

لتعديل بنية جدول نستخدم الصيغة:

```
ALTER TABLE table_name [ADD | DROP COLUMN] (column_name [data_type] );
```

تساعد ADD على إضافة حقول جديدة، في حين تؤدي DROP إلى حذف حقول موجودة.

مثال:

لدينا الجدول Members الذي يحتوي الأرقام التسلسلية للأعضاء ID وأسماء الأعضاء Name.

يراد تعديل بنية الجدول بإضافة حقل جديد لإدخال نمط العضوية Type. لذا نكتب الصيغة:

```
ALTER TABLE Members ADD  (Type varchar(15) );
```

لنفرض أننا نريد إلغاء الحقل ID من بنية الجدول Members. لتنفيذ ذلك نكتب الصيغة:

```
ALTER TABLE Members DROP COLUMN ID;
```

انتبه:
يساعد التعبير ALTER TABLE على إضافة أو إزالة قيود أو فهارس في بنية الجدول وهو ما سنعالجه لاحقاً في هذا الدرس.

لتعديل بنية جدول نستخدم التعبير (ALTER TABLE) الذي يمكننا من إضافة حقول أو حذف حقول من الجدول.

القيود على الحقول

لنعد مجدداً إلى الصيغة الخاصة بتوليد جدول:

```
CREATE TABLE table_name
(column1_name column1_data_type column1_constraints,
column2_name column2_data_type column2_constraints,…);
```

سنلاحظ أننا استخدمنا في هذه الصيغة التعبير column_constraints الذي يمثل القيد المراد تطبيقه على الحقل. سنتعرف فيما يلي على مجموعة من القيود على الحقول ونغطي كل منها بمثال مناسب.

من أهم القيود وأكثرها استخداماً:

- Not Null
- Default
- Primary key
- Unique
- Check
- Identity
- Auto_increment

انتبه:
يمكن لأكثر من قيد أن يطبق على نفس الحقل.

القيود على الحقول

إذا عدنا إلى الصيغة الخاصة بإنشاء جدول سنرى أننا استخدمنا التعبير column_ constraints الذي يمثل القيد المراد تطبيقه على الحقل. سنتعرف فيما يلي على مجموعة من القيود على الحقول ونغطي كل منها بمثال مناسب.

من أهم القيود وأكثرها استخداماً:

- Not Null
- Default
- Primary key
- Unique
- Check
- Identity
- Auto_increment

القيد NOT NULL

يستخدم هذا القيد في حال أردنا أن نمنع إدخال القيمة Null في الحقل المراد تقييده. يكفي لتطبيق هذا القيد إضافة التعبير NOT NULL عند إنشاء الجدول.

مثال:

إذا أردنا إنشاء جدول بأسماء الموظفين وتوصيف عملهم بحيث نمنع إعطاء القيمة Null لأي حقل من حقول الجدول نستخدم الصيغة:

```
CREATE TABLE Employees (name varchar(40) NOT NULL ,
Job varchar(50) NOT NULL);
```

إذا حاولنا تنفيذ الاستعلام التالي بغرض إضافة سجل إلى جدول الموظفين:

```
Insert into Employees(name) values('Adel')
```

سنلاحظ هذا التعبير لن يعمل وسيولد رسالة خطأ تفيد بضرورة إعطاء قيمة للحقل Job، كون القيمة التلقائية التي سيأخذها الحقل Job في حال عدم إدخال أي قيمة له هي القيمة Null.

القيد NOT NULL

يستخدم هذا القيد في حال أردنا أن نمنع إدخال القيمة Null في الحقل المراد تقييده. يكفي لتطبيق هذا القيد إضافة التعبير NOT NULL عند إنشاء الجدول.

تحديد قيمة تلقائية لحقل القيد DEFAULT

من القيود المستخدمة أيضاً بكثرة القيد الخاص بتحديد قيمة تلقائية لحقل ما. يأخذ هذا الحقل تلك القيمة حين لا يتم إسناد أية قيمة بديلة.

نستخدم القيد DEFAULT بالصيغة التالية:

```
CREATE TABLE MyTable
(Column1 varchar(50) DEFAULT 'Unknown' ,
Column2 varchar(10) );
```

في هذه الصيغة سيتم إعطاء القيمة 'Unknown' للحقل Column1 في أي سجل جديد يتم إنشاؤه دون تحديد قيمة لـ Column1.

مثال:

نريد توليد جدول على يحتوي وصف للبضائع (Description) وعدد الأيام الذي يلزم لنقلها إلى المستودع الرئيسي(Days) . نعلم مسبقاً أن عدد الأيام اللازم لعملية النقل هو يومان بصورة عامة.

لتوليد الجدول نكتب الصيغة:

```
CREATE TABLE Shipments
(Description varchar(75) Not Null , Days INT DEFAULT 2 Not Null);
```

نلاحظ أننا أنشأنا الجدول Shipments الذي يحتوي:

- الحقل Description ومنعنا إعطاء القيمة Null لهذا الحقل.
- الحقل Days من نمط بيانات الأعداد الصحيحة وحددنا القيمة 2 كقيمة تلقائية للحقل.

إذا حاولنا إدراج سجل ضمن هذا الجدول دون إعطاء قيمة لـ Days كما في الصيغة:

```
INSERT INTO Shipments (Description) Values ('Computer');
```

سيكون شكل السجل الذي سيتم إدراجه في الجدول هو: 2 | Computer

حيث أخذ الحقل Days في هذا السجل القيمة 2 علماً أننا لم نحدد هذه القيمة في صيغة إدراج السجل.

تحديد قيمة تلقائية لحقل القيد DEFAULT

يُعتبر القيد DEFAULT من القيود المستخدمة أيضاً بكثرة القيد الخاص بتحديد قيمة تلقائية لحقل ما. يأخذ هذا الحقل تلك القيمة حين لا يتم إسناد أية قيمة بديلة.

القيد PRIMARY KEY

من أهم الخصائص التي تميز الجداول في قواعد البيانات العلائقية والتي تعتبر من الشروط الأساسية للنموذج العلائقي لـ Codd، ضرورة وجود حقل في كل جدول يلعب قيمه دور مميز لكل سجل من السجلات.

تعمل قيمة هذا المفتاح على تمييز كل سجل من سجلات الجدول بصورة وحيدة ويسمى المفتاح الرئيسي للجدول. لهذا الغرض يُستخدم القيد PRIMARY KEY ليحدد أي الحقول هو حقل مفتاح رئيسي لجدول ما.

يستخدم هذا القيد الصيغة:

```
CREATE TABLE MyTable
(Column1 data_type Not Null , Column2 data_type ,
Constraint myPrimaryKey PRIMARY KEY (Column1));
```

أنشأنا هنا قيد باسم MyPrimaryKey وقيدنا به الحقل Column1 بحيث يجب أن تكون قيمة هذا الحقل وحيدة وتميز كل سجل.

إذا لم نرد تحديد اسم لهذا القيد يمكننا كتابة الصيغة:

```
CREATE TABLE MyTable
(Column1 data_type Not Null , Column2 data_type ,
PRIMARY KEY (Column1));
```

أو بصورة أبسط يمكننا استخدام الصيغة:

```
CREATE TABLE MyTable
(Column1 data_type PRIMARY KEY Not Null , Column2 data_type ,
PRIMARY KEY (Column1));
```

مثال:

نريد إنشاء جدول CreditCards لتخزين أرقام البطاقات الائتمانية cardNumber وأسماء أصحابها cardHolder.

نعلم هنا أن أرقام البطاقات الائتمانية فريدة ولا تتكرر لذا سنعتمدها كمفتاح رئيسي في جدولنا وسنستخدم الصيغة المبسطة لقيد المفتاح الرئيسي:

```
CREATE TABLE CreditCards
(cardNumber varchar(20) PRIMARY KEY Not Null ,
cardHolder varchar(50) Not Null);
```

القيد Primary Key

من أهم الخصائص التي تميز الجداول في قواعد البيانات العلائقية والتي تعتبر من الشروط الأساسية للنموذج العلائقي لـ Codd، ضرورة وجود حقل في كل جدول يلعب قيمه دور مميز لكل سجل من السجلات.

تعمل قيمة هذا المفتاح على تمييز كل سجل من سجلات الجدول بصورة وحيدة ويسمى المفتاح الرئيسي للجدول. لهذا الغرض يُستخدم القيد PRIMARY KEY ليحدد أي الحقول هو حقل مفتاح رئيسي لجدول ما.

القيد UNIQUE

يُستخدم القيد UNIQUE لمنع تكرار قيمة حقل في أكثر من سجل، ولكن هذا لايعني أن هذا الحقل سيصبح المفتاح الرئيسي للجدول.

الصيغة المستخدمة لهذا القيد هي كالتالي:

```
CREATE TABLE MYTable
(Column1 data_type UNIQUE , Column2 data_type);
```

مثال:

إذا أردنا توليد جدول PhoneBook بأسماء الأشخاص Name وأرقام هواتفهم Phone بحيث يكون رقم الهاتف مفتاح رئيسي واسم الشخص **ذو قيمة وحيدة لا تتكرر** نستخدم الصيغة:

```
CREATE TABLE PhoneBook
(Name varchar(50) UNIQUE , Phone Primary Key Not Null);
```

القيد UNIQUE

يستخدم القيد UNIQUE لمنع تكرار قيمة حقل في أكثر من سجل، ولكن هذا لايعني أن هذا الحقل سيصبح المفتاح الرئيسي للجدول.

القيد Check

يعتبر القيد Check من أكثر القيود مرونة لأنه يسمح لنا باستخدام طيف واسع من الشروط على القيم التي يتم إدراجها ضمن الجدول.

تشبه طريقة استخدام شروط القيد Check لطريقة استخدام التعبير Where حيث يتم إعادة تقييم الشرط في كل مرة نضيف فيها سجلاً جديداً أو نعدل سجل موجود.

تكون صيغة استخدام القيد Check كالتالي:

```
CREATE TABLE MyTable
(Column1 data_type ,
Column2 data_type ,
Constraint Cname CHECK (Condition));
```

حيث تعبر Cname عن اسم القيد وتعبر Condition عن شرط يُستخدم فيه اسم الحقل المراد تقييده بـ Check.

مثال:

نريد إنشاء جدول Ages يتضمن أسماء Name وأعمارAge مجموعة من الأطفال تتراوح بين سنة و12 سنة.

نكتب الصيغة:

```
CREATE TABLE Ages
(Name varchar(50) Not Null ,
Age INT ,
Constraint CheckAge CHECK (Age between 1 And 12));
```

ويمكن أن نكتب صيغة تحقق نفس الغرض مع حذف اسم القيد CheckAge إذا كنا لانريد اسم للقيد.

```
CREATE TABLE Ages
(Name varchar(50) Not Null ,
Age INT CHECK(Age between 1 And 12));
```

يمكن لشرط القيد Check أن يتألف من تعبيرات منطقية تحتوي على عمليات منطقية مثل And أو Or فمثلاً إذا أردنا السماح بإدخال عمر بين 1 و12 أو يساوي 15 نكتب الصيغة:

```
CREATE TABLE Ages
(Name varchar(50) Not Null ,
Age INT CHECK(Age = 15 OR Age between 1 And 12));
```

يمكن وضع أكثر من قيد Check على حقل وحيد فمثلاً إذا أردنا السماح بالقيم للعمر بين 1 و12 عدا العمر 3 نكتب الصيغة:

```
CREATE TABLE Ages
(Name varchar(50) Not Null ,
Age INT,
Constraint CheckAge1 CHECK (Age between 1 And 12),
Constraint CheckAge2 CHECK (Age <> 3));
```

انتبه:

لا يطبق القيد Check عندما لايتلقى الحقل أي إدخالات (أي عندما تكون قيمته Null).

القيد Check

يعتبر القيد Check من أكثر القيود مرونة لأنه يسمح لنا باستخدام طيف واسع من الشروط على القيم التي يتم إدراجها ضمن الجدول.

- تشبه طريقة استخدام القيد Check طريقة استخدام التعبير Where حيث يتم إعادة تقييم الشرط في كل مرة نضيف فيها سجلاً جديداً أو نعدل سجل موجود.
- يمكن لشرط القيد Check أن يتألف من تعبيرات منطقية تحتوي على عمليات منطقية مثل And أو Or.
- يمكن وضع أكثر من قيد Check على حقل وحيد.

القيد IDENTITY وAUTO_INCREMENT

توفر قواعد البيانات آلية لتوليد قيم عددية بصورة آلية كقيم لحقل ما عند إضافة سجل جديد إلى الجدول. يمكن استخدام هذه التقنية بالاشتراك مع القيد PRIMARY KEY لتوليد قيم تسلسلية تلقائية في حقل يكون هو حقل المفتاح الرئيسي للجدول.

تستخدم قواعد البيانات تعابير مختلفة لتخديم نفس الغرض:

- نستخدم في SQL Server التعبير IDENTITY
- نستخدم في MySQL التعبير AUTO_INCREMENT
- نستخدم في DB2 التعبير GENRATED ALWAYS AS IDENTITY
- نستخدم في Access التعبير AUTOINCREMENT
- أما في Oracle فنحتاج لإنشاء متتالية بالأمر Create Sequence

مثال:

نود إنشاء جدول بأرقام تسلسلية للطلاب وأسمائهم:

- في قاعدة بيانات SQL Server:

الأرقام تبدأ من 100 وتتزايد بمقدار 1:

```
CREATE TABLE Students
(Name varchar(50) ,
ID INT IDENTITY (100,1) PRIMARY KEY NOT NULL);
```

في حال لم نحدد البذرة لبدء العد (100) والتزايد (1) تكون القيمة التلقائية للبذرة والتزايد هي (1).

- تصبح الصيغة في حالة MySQL بالشكل:

```
CREATE TABLE Students
(Name varchar(50) ,
ID INT AUTO_INCREMENT PRIMARY KEY NOT NULL);
```

- وتصبح الصيغة في Access:

```
CREATE TABLE Students
(Name varchar(50) ,
ID INT AUTOINCREMENT (100,1) PRIMARY KEY NOT NULL);
```

- أما في DB2 فتصبح الصيغة:

```
CREATE TABLE Students
(Name varchar(50) ,
ID INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY);
```

في هذه الصيغة نستخدم ALWAYS للإشارة إلى أننا لن نسمح بإدخال قيم يدوية لحقل التعداد الآلي. لتمكين إدخال القيم يدوياً يمكننا استخدام BY DEFAULT عوضاً عن ALWAYS.

القيد IDENTITY , AUTO_INCREMENT

توفر قواعد البيانات آلية لتوليد قيم عددية بصورة آلية كقيم لحقل ما عند إضافة سجل جديد إلى الجدول. يمكن استخدام هذه التقنية بالاشتراك مع القيد PRIMARY KEY لتوليد قيم تسلسلية تلقائية في حقل يكون هو حقل المفتاح الرئيسي للجدول.
تستخدم قواعد البيانات تعابير مختلفة لتخديم نفس الغرض:

- نستخدم في SQL Server التعبير IDENTITY
- نستخدم في MySQL التعبير AUTO_INCREMENT
- نستخدم في DB2 التعبير GENRATED ALWAYS AS IDENTITY
- نستخدم في Access التعبير AUTOINCREMENT
- أما في Oracle فنحتاج لإنشاء متتالية بالأمر Create Sequence

القيد IDENTITY وAUTO_INCREMENT

لا تدعم قواعد بيانات Oracle التعابير IDENTITY أو AUTO_INCREMENT المستخدمة في قواعد البيانات الأخرى. لذا تستخدم Oracle تعبير خاص لإنشاء متوالية هو التعبير CREATE SEQUENCE لمحاكاة عمل هذه القيود. تكون صيغة هذا التعبير على الشكل:

```
CREATE SEQUENCE sequence_name
INCREMENT increment_step
START WITH start_seed;
```

- عند إنشاء مثل هذه المتوالية يمكن استخدام القيم التي تأخذها لإدراجها في صيغ SQL أخرى.
- لإعادة قيمة المتوالية وزيادة عداد المتوالية بمقدار الخطوة يكفي استخدام التعبير sequence_name.NextVal

يتم استخدام قيم هذه المتوالية مباشرة في تعبير Insert عند إدراج سجل جديد في جدول ما وذلك بالصيغة:

```
INSERT INTO mytable
(Column1, Column2, Column3)
Values (sequence_name.NextVal, Value2, Value3);
```

نلاحظ هنا أننا أعطينا للحقل Column1 القيمة الخاصة بالمتوالية واستخدمنا الطريقة NextVal لاستجرار القيمة التالية للمتوالية.

مثال:

نريد إنشاء جدول Products في قاعدة بيانات Oracle يحتوي حقلين الأول خاص بأرقام المنتجات productID والثاني وصف المنتجات ProductDescription. كما نود جعل الحقل productID حقل مفتاح رئيسي وجعل قاعدة البيانات تعطي لهذا الحقل أرقام متسلسلة تلقائية.

لإتمام هذا العمل نقوم أولاً بإنشاء الجدول بالصيغة:

```
CREATE TABLE Products
(productID INT PRIMARY KEY NOT NULL ,
productDescription varchar(75));
```

ثم نقوم بإنشاء متوالية وليكن اسمها مثلاً Counter وذلك بالصيغة:

```
CREATE SEQUENCE Counter;
```

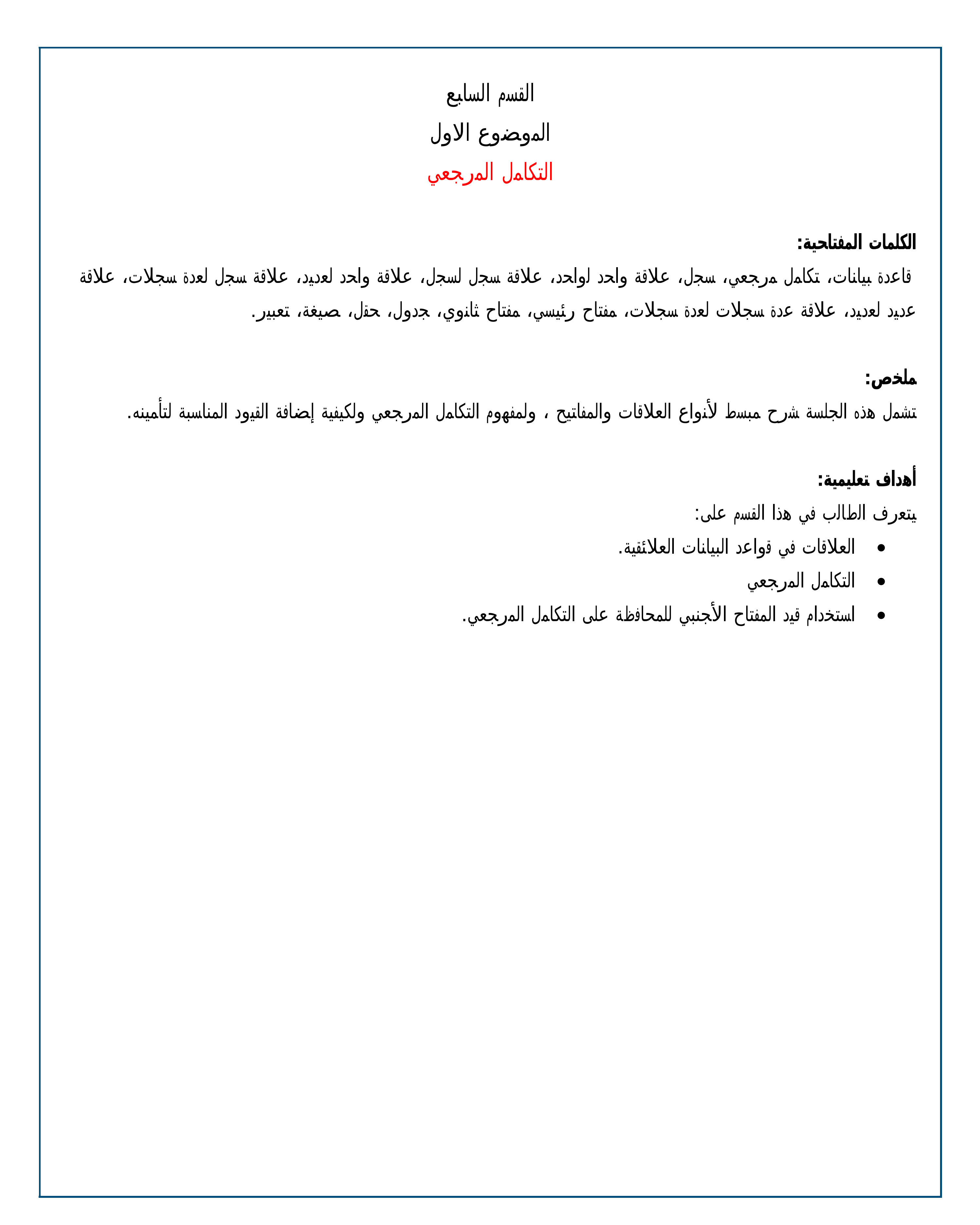
حيث لم نحدد هنا بداية العد والخطوة لذلك ستحدد قيمتها تلقائياً بالعدد 1.

عند إدراج سجل في جدولنا الجديد يجب استخدام المتوالية كقيمة للحقل productID وذلك بالشكل:

```
INSERT INTO Products
(productID , productDescription)
Values (Counter.NextVal , ‘any Porduct description’);
```

لا تدعم قواعد بيانات Oracle التعابير IDENTITY أو AUTO_INCREMENT المستخدمة في قواعد البيانات الأخرى. لذا تستخدم Oracle تعبير خاص لإنشاء متوالية هو التعبير CREATE SEQUENCE لمحاكاة عمل هذه القيود.

عند إنشاء مثل هذه المتوالية يمكن استخدام القيم التي تأخذها لإدراجها في صيغ SQL أخرى.
لإعادة قيمة المتوالية وزيادة عداد المتوالية بمقدار خطوة يكفي استخدام التعبير sequence_name.NextVal



القسم السابع

الموضوع الاول

التكامل المرجعي

الكلمات المفتاحية:

قاعدة بيانات، تكامل مرجعي، سجل، علاقة واحد لواحد، علاقة سجل لسجل، علاقة واحد لعديد، علاقة سجل لعدة سجلات، علاقة عديد لعديد، علاقة عدة سجلات لعدة سجلات، مفتاح رئيسي، مفتاح ثانوي، جدول، حقل، صيغة، تعبير.

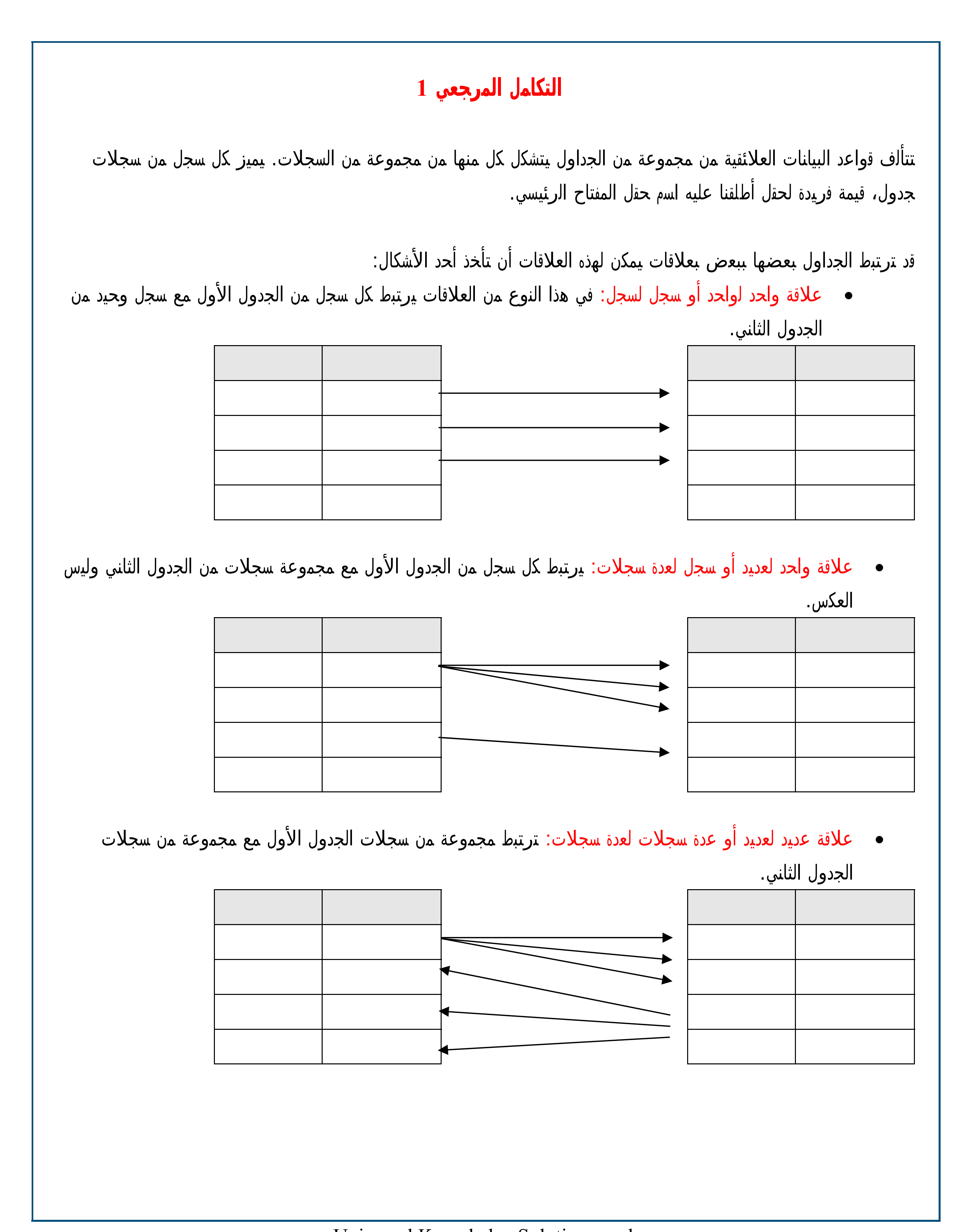
ملخص:

تشمل هذه الجلسة شرح مبسط لأنواع العلاقات والمفاتيح ، ولمفهوم التكامل المرجعي ولكيفية إضافة القيود المناسبة لتأمينه.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- العلاقات في قواعد البيانات العلائقية.
- التكامل المرجعي
- استخدام قيد المفتاح الأجنبي للمحافظة على التكامل المرجعي.



التكامل المرجعي 1

تتألف قواعد البيانات العلائقية من مجموعة من الجداول يتشكل كل منها من مجموعة من السجلات. يميز كل سجل من سجلات جدول، قيمة فريدة لحقل أطلقنا عليه اسم حقل المفتاح الرئيسي.

قد ترتبط الجداول بعضها ببعض بعلاقات يمكن لهذه العلاقات أن تأخذ أحد الأشكال:

- علاقة واحد لواحد أو سجل لسجل: في هذا النوع من العلاقات يرتبط كل سجل من الجدول الأول مع سجل وحيد من الجدول الثاني.

- علاقة واحد لعديد أو سجل لعدة سجلات: يرتبط كل سجل من الجدول الأول مع مجموعة سجلات من الجدول الثاني وليس العكس.

- علاقة عديد لعديد أو عدة سجلات لعدة سجلات: ترتبط مجموعة من سجلات الجدول الأول مع مجموعة من سجلات الجدول الثاني.

التكامل المرجعي 1

تتألف قواعد البيانات العلائقية من مجموعة من الجداول يتشكل كل من هذه الجداول من مجموعة من السجلات. يميز كل سجل من سجلات جدول قيمة فريدة لحقل أطلقنا عليه اسم حقل المفتاح الرئيسي.

قد ترتبط الجداول بعضها ببعض بعلاقات يمكن لهذه العلاقات أن تأخذ أحد الأشكال:

- علاقة واحد لواحد أو سجل لسجل: في هذا النوع من العلاقات يرتبط كل سجل من الجدول الأول مع سجل وحيد من الجدول الثاني.
- علاقة واحد لعديد أو سجل لعدة سجلات: يرتبط كل سجل من الجدول الأول مع مجموعة سجلات من الجدول الثاني وليس العكس.
- علاقة عديد لعديد أو عدة سجلات لعدة سجلات: ترتبط مجموعة من سجلات الجدول الأول مع مجموعة من سجلات الجدول الثاني.
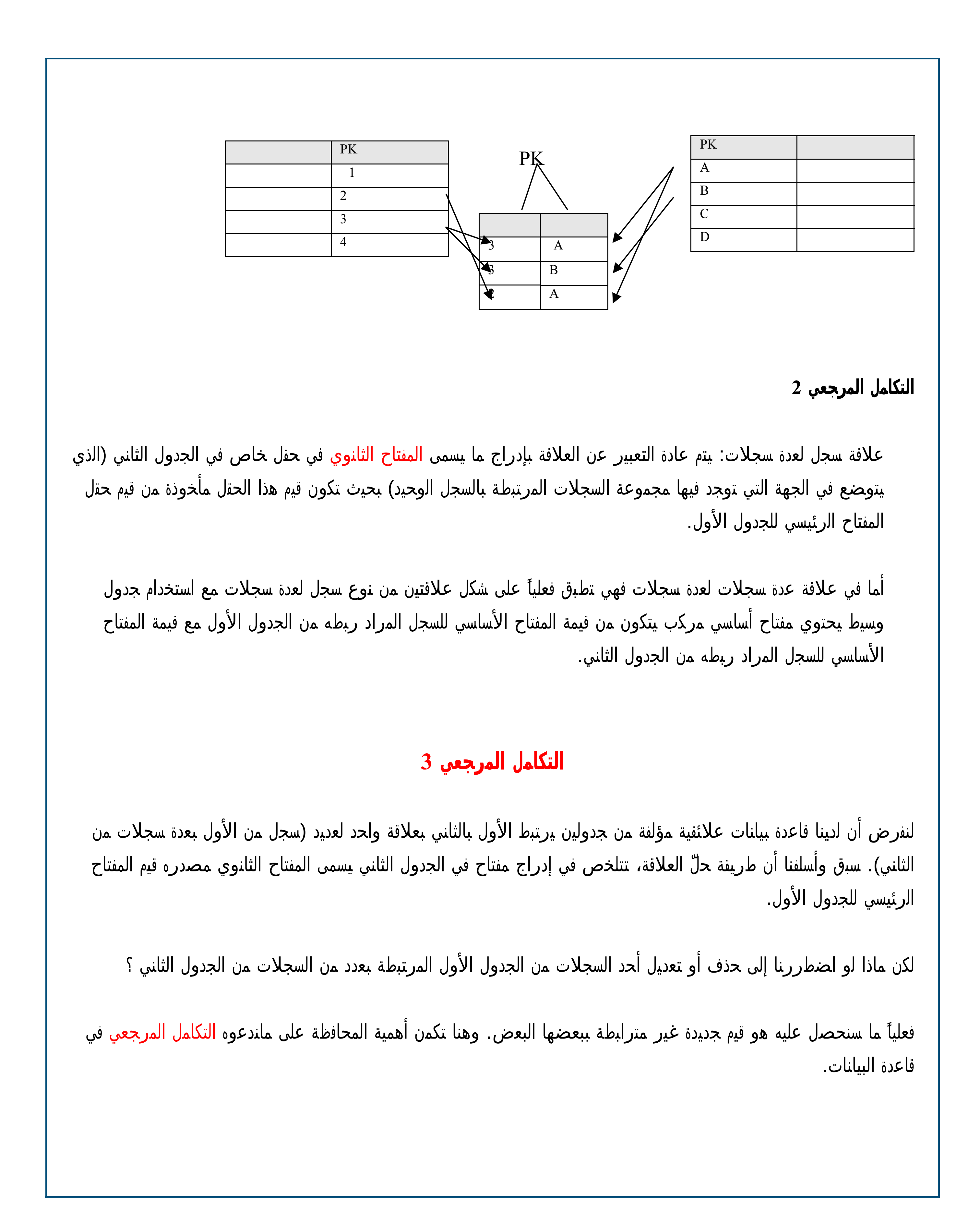
التكامل المرجعي 2

علاقة سجل لعدة سجلات: يتم عادة التعبير عن العلاقة بإدراج ما يسمى المفتاح الثانوي في حقل خاص في الجدول الثاني (الذي يتوضع في الجهة التي توجد فيها مجموعة السجلات المرتبطة بالسجل الوحيد) وبحيث تكون قيم هذا الحقل مأخوذة من قيم حقل المفتاح الرئيسي للجدول الأول.

	PK
	2
	5
	3
	8

FK	
2	
2	
2	
3	

أما في علاقة عدة سجلات لعدة سجلات فهي تطبق فعلياً على شكل علاقتين من نوع سجل لعدة سجلات مع استخدام جدول وسيط يحتوي مفتاح أساسي مركب يتكون من قيمة المفتاح الأساسي للسجل المراد ربطه من الجدول الأول مع قيمة المفتاح الأساسي للسجل المراد ربطه من الجدول الثاني.



التكامل المرجعي 2

علاقة سجل لعدة سجلات: يتم عادة التعبير عن العلاقة بإدراج ما يسمى المفتاح الثانوي في حقل خاص في الجدول الثاني (الذي يتوضع في الجهة التي توجد فيها مجموعة السجلات المرتبطة بالسجل الوحيد) بحيث تكون قيم هذا الحقل مأخوذة من قيم حقل المفتاح الرئيسي للجدول الأول.

أما في علاقة عدة سجلات لعدة سجلات فهي تطبق فعلياً على شكل علاقتين من نوع سجل لعدة سجلات مع استخدام جدول وسيط يحتوي مفتاح أساسي مركب يتكون من قيمة المفتاح الأساسي للسجل المراد ربطه من الجدول الأول مع قيمة المفتاح الأساسي للسجل المراد ربطه من الجدول الثاني.

التكامل المرجعي 3

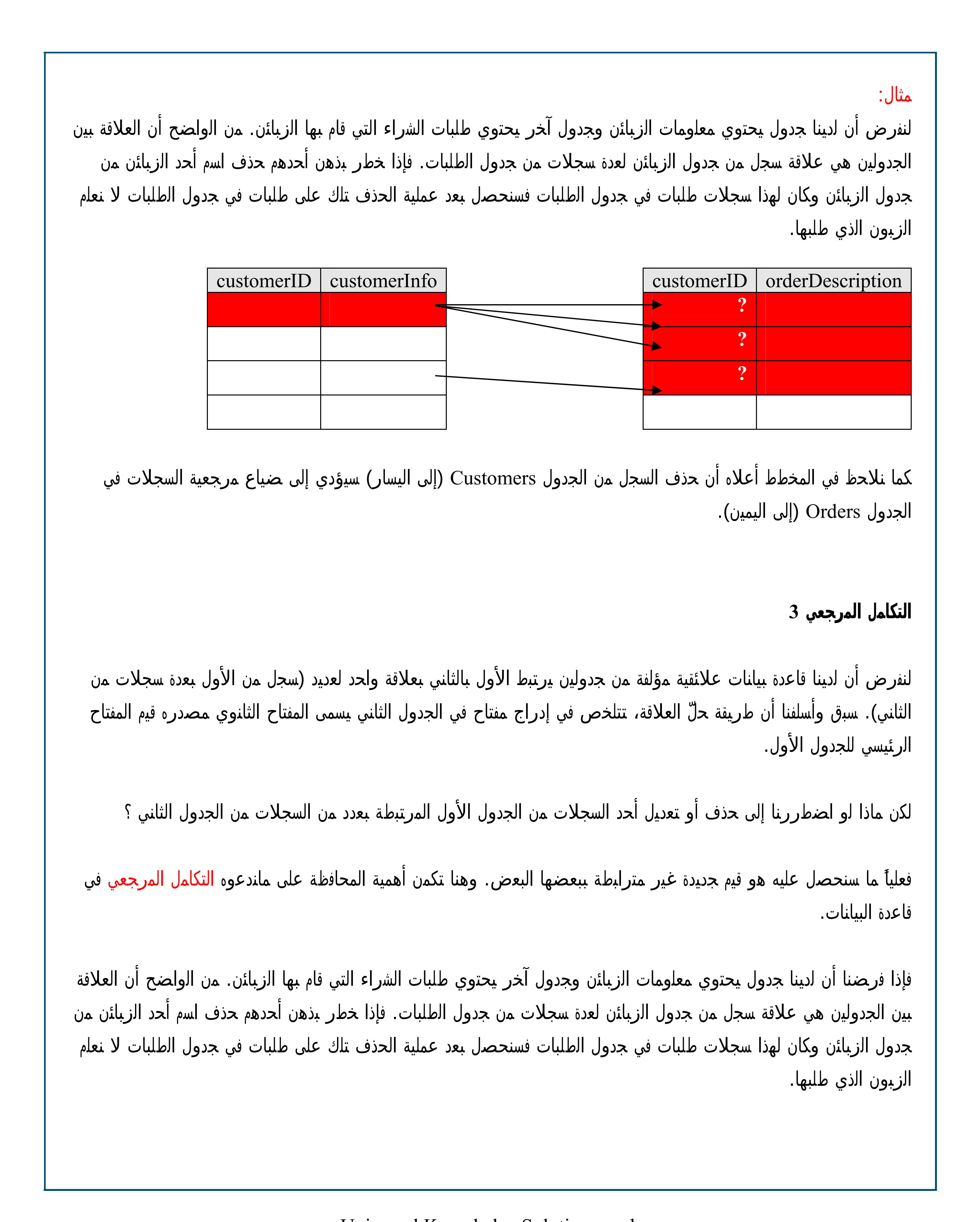
لنفرض أن لدينا قاعدة بيانات علائقية مؤلفة من جدولين يرتبط الأول بالثاني بعلاقة واحد لعديد (سجل من الأول بعدة سجلات من الثاني). سبق وأسلفنا أن طريقة حلّ العلاقة، تتلخص في إدراج مفتاح في الجدول الثاني يسمى المفتاح الثانوي مصدره قيم المفتاح الرئيسي للجدول الأول.

لكن ماذا لو اضطررنا إلى حذف أو تعديل أحد السجلات من الجدول الأول المرتبطة بعدد من السجلات من الجدول الثاني ؟

فعلياً ما سنحصل عليه هو قيم جديدة غير مترابطة ببعضها البعض. وهنا تكمن أهمية المحافظة على ماندعوه التكامل المرجعي في قاعدة البيانات.

مثال:

لنفرض أن لدينا جدول يحتوي معلومات الزبائن وجدول آخر يحتوي طلبات الشراء التي قام بها الزبائن. من الواضح أن العلاقة بين الجدولين هي علاقة سجل من جدول الزبائن لعدة سجلات من جدول الطلبات. فإذا خطر بذهن أحدهم حذف اسم أحد الزبائن من جدول الزبائن وكان لهذا سجلات طلبات في جدول الطلبات فسنحصل بعد عملية الحذف تلك على طلبات في جدول الطلبات لا نعلم الزبون الذي طلبها.



كما نلاحظ في المخطط أعلاه أن حذف السجل من الجدول Customers (إلى اليسار) سيؤدي إلى ضياع مرجعية السجلات في الجدول Orders (إلى اليمين).

التكامل المرجعي 3

لنفرض أن لدينا قاعدة بيانات علائقية مؤلفة من جدولين يرتبط الأول بالثاني بعلاقة واحد لعديد (سجل من الأول بعدة سجلات من الثاني). سبق وأسلفنا أن طريقة حلّ العلاقة، تتلخص في إدراج مفتاح في الجدول الثاني يسمى المفتاح الثانوي مصدره قيم المفتاح الرئيسي للجدول الأول.

لكن ماذا لو اضطررنا إلى حذف أو تعديل أحد السجلات من الجدول الأول المرتبطة بعدد من السجلات من الجدول الثاني ؟

فعلياً ما سنحصل عليه هو قيم جديدة غير مترابطة ببعضها البعض. وهنا تكمن أهمية المحافظة على ماندعوه التكامل المرجعي في قاعدة البيانات.

فإذا فرضنا أن لدينا جدول يحتوي معلومات الزبائن وجدول آخر يحتوي طلبات الشراء التي قام بها الزبائن. من الواضح أن العلاقة بين الجدولين هي علاقة سجل من جدول الزبائن لعدة سجلات من جدول الطلبات. فإذا خطر بذهن أحدهم حذف اسم أحد الزبائن من جدول الزبائن وكان لهذا سجلات طلبات في جدول الطلبات فسنحصل بعد عملية الحذف تلك على طلبات في جدول الطلبات لا نعلم الزبون الذي طلبها.

التكامل المرجعي 4

لضمان عدم كسر قاعدة التكامل المرجعي في SQL، علينا استخدام قيد يعرف أحد الحقول في جدول على أنه حقل مفتاح ثانوي لجدول آخر بحيث يفشل أي تعبير لا يحترم قاعدة التكامل المرجعي بعد استخدام هذا القيد.

يتم لهذا الغرض استخدام الصيغة:

```
CREATE TABLE myTable
(Column1 Column1Type PRIMARY KEY NOT NULL , Column2 Column2Type ,
Column3 Column3Type ,
CONSTAINT foreign_key_name FOREIGN KEY (Column3)
REFERENCES other_table (other_table_primary_key));
```

نلاحظ في الصيغة السابقة أننا أنشأنا الجدول myTable الذي يحوي الحقول Column1 كحقل مفتاح رئيسي والحقل Column2 والحقل Column3 الذي تم تقييده بالقيد FOREIGN KEY لجعله مفتاح ثانوي مرتبط بالمفتاح الرئيسي للجدول other_table والذي يحمل اسم other_table_primary_key.

مثال:

إذا أردنا إنشاء الجدول Brands الذي يحتوي أسماء وأرقام ماركات أجهزة الحواسب المتوفرة بمتجر، والجدول Models الذي يحتوي الموديلات المتوفرة من كل ماركة من الماركات، وأردنا ربط الجدول Brands مع الجدول Models بعلاقة سجل من جدول Brands إلى عدة سجلات من جدول Models مع ضمان التكامل المرجعي نكتب الصيغة التالية لإنشاء الجدول Brands:

```
CREATE TABLE Brands (
brandID INT PRIMARY KEY NOT NULL ,
brandName varchar (50));
```

لإنشاء الجدول Models نستخدم الصيغة:

```
CREATE TABLE Models (
modelID INT PRIMARY KEY NOT NULL ,
modelName varchar (50) ,
modelBrand INT ,
CONSTRAINT myFK FOREIGN KEY (modelBrand)
REFERENCES Brands (brandID));
```

هذه الصيغة صالحة في SQL Server, Oracle, DB2 وAccess.

يمكن كتابة هذه الصيغة بشكل أكثر اختصاراً إذا كنا لا ننوي تحديد اسم القيد بالشكل:

```
CREATE TABLE Models (
modelID INT PRIMARY KEY NOT NULL ,
modelName varchar (50) ,
modelBrand INT ,
FOREIGN KEY (modelBrand)
REFERENCES Brands (brandID));
```

التكامل المرجعي 4

لضمان عدم كسر قاعدة التكامل المرجعي في SQL، علينا استخدام قيد يعرف أحد الحقول في جدول على أنه حقل مفتاح ثانوي لجدول آخر بحيث يفشل أي تعبير لا يحترم قاعدة التكامل المرجعي بعد استخدام هذا القيد.

التكامل المرجعي في MySQL

في MySQL تختلف الصيغة بنقطتين أساسيتين:

الأولى هي أن الجداول الداخلة في العلاقة يجب أن تكون من نوع الجداول الخاص في قواعد بيانات MySQL وهو InnoDB، والثانية هي ضرورة تعريف حقل المفتاح الثانوي كفهرس فتصبح الصيغة من الشكل:

```
CREATE TABLE myTable
(Column1 Column1Type PRIMARY KEY NOT NULL , Column2 Column2Type ,
Column3 Column3Type ,
FOREIGN KEY (Column3)
REFERENCES other_table (other_table_primary_key)
INDEX myIndex (Column3))
Type = InnoDB;
```

التكامل المرجعي في MySQL

في MySQL تختلف الصيغة بنقطتين أساسيتين:

الأولى هي أن الجداول الداخلة في العلاقة يجب أن تكون من نوع الجداول الخاص في قواعد بيانات MySQL وهو InnoDB
والثانية هي ضرورة تعريف حقل المفتاح الثانوي كفهرس.



القسم السابع

الموضوع الثاني

استخدام المناظير والجداول المؤقتة والفهارس في قواعد بيانات العلائقية

الكلمات المفتاحية:

قاعدة بيانات، إنشاء، حذف، سجل، قيمة، قيد، مفتاح رئيسي، جدول، حقل، صيغة، تعبير، فهرس، منظار.

ملخص:

تشمل هذه الجلسة تغطية المعلومات المتعلقة باستخدام المنظار والجداول المؤقتة والفهارس في قواعد بيانات العلائقية.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- كيفية إنشاء وحذف وتعديل بنية منظار.
- كيفية إنشاء وحذف الجداول المؤقتة.
- كيفية استخدام الفهارس وإنشاءها وإزالتها.

المنظار في SQL

استخدمنا حتى الآن الكثير من صيغ SQL المعقدة نسبياً للاستعلام عن البيانات. لكن، سيصبح من المرهق كتابة هذه الصيغ في كل مرة إذا كانت هذه العمليات تُستخدم بكثرة. لهذا السبب، تتضمن SQL تعابير خاصة تمكننا من توليد جدول افتراضي ندعوه منظار.

تساعد المناظير في استعادة البيانات التي يُرجعها استعلام، فهي توفر جداول افتراضية تحتوي على البيانات ضمن التشكيل المطلوب.

يمكن معاملة المنظار كأي جدول من جداول قواعد البيانات. إذ يكمن الفرق الوحيد بينه وبين الجداول في كون البيانات التي تحتويه مخزنة في جداول أخرى.

تساعد المناظير في منع المستخدمين من الوصول إلى الجداول الأصلية في قواعد البيانات. إذ يحتوي المنظار على جزءٍ من البيانات المتوفرة في الجدول الأصلي.

لا يمكن عادةً، تعديل البيانات من خلال منظار، فهو يشكل نسخة بيانات قابلة للقراءة فقط يمكن تخصيصها للمستخدمين ذوي الصلاحيات المنخفضة.

إنشاء منظار

لإنشاء منظار نستخدم الصيغة:

```
CREATE VIEW view_name AS query;
```

حيث view_name هو اسم المنظار وquery هو استعلام يمكن أن نستخدم فيه أي من التقنيات التي تعرفنا عليها مسبقاً.

مثال:

نفترض أن لدينا استعلاماً معقداً نسبياً يعيد قيم من جدولين مرتبطين بأسلوب Inner Join وأردنا إنشاء منظار ليحتوي قيم الحقول المعادة من هذا الاستعلام نكتب الصيغة:

```
CREATE VIEW MySimpleView Projects.projectName ,
count (Tasks.taskID) AS TasksNumber
From Tasks Inner Join Projects
ON Tasks.projectID = Projects.projectID
Group by projectName;
```

هنا قمنا بإنشاء منظار باسم MySimpleView يحتوي الحقول projectName وtasksNumber.

بعد إنشاء هذا المنظار يمكننا ببساطة استخدام أي استعلام والتعامل مع المنظار على أنه من خلال الصيغة التالية مثلاً:

```
Select projectName from MySimpleView;
```

تعديل البيانات عبر تعديل المنظار

عادة ليس من الممكن تعديل البيانات عبر تطبيق التعابير Insert, Update وDelete على المنظار ولكن يمكن تحقيق هذا الغرض في حال توفرت الشروط التالية:

- يجب ألا يحوي الاستعلام الخاص بالمنظار أي تابع تجميعي كما يجب ألا يستخدم تعبير Group By.
- يجب ألا يحوي الاستعلام الخاص بالمنظار التعبير Top أو Distinct.
- يجب ألا يحوي الاستعلام الخاص بالمنظار حقول تم إجراء عمليات حسابية عليها أو تم حسابها.

تعديل بنية منظار

لتعديل بنية منظار ما يمكننا تغيير توصيفه باستخدام الصيغة التالية:

```
ALTER VIEW viewName AS newQuery;
```

تعمل الصيغة السابقة في SQL Server أما في Oracle فتصبح الصيغة:

```
CREATE OR REPLACE VIEW viewName AS newQuery
```

انتبه:

- يستخدم التعبير ALTER VIEW في Oracle وDB2 لأداء أغراض مختلفة عن تلك المستخدمة في SQL Server.
- أفضل طريقة لتعديل منظار في DB2 هي حذفه وإعادة إنشائه بالشكل المطلوب من جديد.
- في MySQL لم تكن المعاينات مدعومة قبل النسخة 5.0.1 ولكن تم إضافتها وهي تستخدم صيغة مشابهة للصيغة المستخدمة في SQL Server.

مثال:

إذا كان لدينا الجدول Studio الحاوي على أسماء وأرقام الاستوديوهات studioName, studioNumber وأرقام البرامج programNumber التي ستصور في مركز تلفزيوني. وإذا كان لدينا الجدول Actors الحاوي على أسماء الممثلين ActorName وأسماء البرامج التي يشتركون في تمثيلها programName. وإذا أردنا تعديل بنية المنظار ActorsStudios في SQL Server وMySQL تصبح الصيغة:

```
ALTER VIEW  ActorsStudios AS
Select actorName , studioName from Actors Inner Join Studios
ON Studios.studioNumber = Actors.studioNumber;
```

ويصبح المثال في Oracle:

```
CREATE OR REPLACE VIEW  ActorsStudios AS
Select actorName , studioName from Actors Inner Join Studios
ON Studios.studioNumber = Actors.studioNumber;
```

حذف منظار

يمكننا حذف منظار بصورة كاملة من قاعدة البيانات باستخدام التعبير DROP VIEW.

إن حذف منظار لا يعني حذف الأصول أو الجداول الأساسية التي دخلت في الاستعلام الذي أنشأ المنظار.

يستخدم DROP VIEW بالصيغة:

```
DROP VIEW viewName;
```

مثال:

فإذا كان لدينا المنظار MyView الذي أنشأناه بالصيغة:

```
CREATE VIEW MyView AS
Select * from MyTable;
```

لحذف المنظار MyView نستخدم:

```
DROP VIEW  MyView;
```

بعد تنفيذ هذه الصيغة سيتم حذف المنظار MyView وستفشل الاستعلامات على هذا المنظار مثل الاستعلام:

```
Select* from MyView;
```

كما ذكرنا أعلاه أن حذف منظار لا يعني حذف الأصول أو الجداول الأساسية التي دخلت في الاستعلام الذي أنشأ المرآى ففي مثالنا، لا يتأثر الجدول MyTable بحذف المنظار MyView.

حذف مرآى

يمكننا حذف منظار بصورة كاملة من قاعدة البيانات باستخدام التعبير DROP VIEW. إن حذف منظار لا يعني حذف الأصول أو الجداول الأساسية التي دخلت في الاستعلام الذي أنشأ المنظار.

الجداول المؤقتة

نُعرف الجداول المؤقتة على أنها الجداول التي يتم إنشاؤها لتخديم غرض مؤقت ويستمر وجودها لفترة محددة من الزمن أو خلال جلسة ما أو خلال مناقلة.

استخدام الجداول المؤقتة في SQL Server:

تختلف الصيغة المخصصة لإنشاء جدول مؤقت في قواعد البيانات المختلفة ففي حالة SQL Server تكون على الشكل:

```
CREATE TABLE # tmp_Table (Field1Name Field1Type , Field2Name
Field2Type;
```

نلاحظ هنا أننا استخدمنا عبارة إنشاء الجدول المألوفة CREATE TABLE ولكننا أضفنا إشارة # قبل اسم الجدول للتعبير عن كونه جدول مؤقت.

تُستخدم في هذه الحالة (حالة SQL Server) إشارة # للدلالة على كون الجدول المؤقت محلي أي لا يمكن الوصول إليه إلا عبر الاتصال الذي قام بإنشائه. وتستخدم الإشارة ## للتعبير عن كون الجدول المؤقت عام أي يمكن الوصول إليه عبر أي اتصال.

يتم إنشاء الجداول المؤقتة في SQL Server دائماً في قاعدة البيانات tempdb الخاصة بالنظام.

عندما لا يحذف إتصال ما الجداول التي أنشائها، يتم حذفها تلقائياً فور إلغاء الاتصال.

مثال:

لدينا الجدول Students الذي يحتوي قائمة بأسماء الطلاب studentName وأرقامهم studentID وعلاماتهم studentMark.

لإنشاء الجدول المؤقت الذي يحتوي أسماء الطلاب ومعدلاتهم نستخدم الصيغة:

```
CREATE TABLE #tmp (studentName varchar(50) , average INT);
```

لملء هذا الجدول يمكننا استخدام الصيغة:

```
Insert Into #tmp select Students.studentName AS studentName ,
AVG (studentMark) AS average from Students;
```

وللاستعلام مثلاً عن الطلاب الراسبين أي الذين حصلوا على معدلات أقل من 50 يمكننا كتابة الاستعلام:

```
Select studentName , average from #temp where average<50;
```

الجداول المؤقت

نُعرف الجداول المؤقتة على أنها الجداول التي يتم إنشاؤها لتخديم غرض مؤقت ويستمر وجودها لفترة محددة من الزمن أو خلال جلسة ما أو خلال مناقلة.

تختلف الصيغة المخصصة لإنشاء جدول مؤقت في قواعد البيانات المختلفة.

استخدام الجداول المؤقتة في SQL Server:

يتم إنشاء الجداول المؤقتة في SQL Server دائماً في قاعدة البيانات tempdb الخاصة بالنظام. وعندما لا يحذف إتصال ما الجداول التي أنشائها، يتم حذفها تلقائياً فور إلغاء الاتصال.

الجداول المؤقتة

استخدام الجداول المؤقتة في Oracle:

تستخدم قواعد بيانات Oracle لإنشاء جدول مؤقت التعبير:

CREATE GLOBAL TEMPORARY TABLE عوضاً عن CREATE TABLE المستخدم في SQL Server.

ويأخذ الصيغة:

```
CREATE GLOBAL TEMPORARY TABLE temp_table AS query;
```

تعبر query عن الاستعلام الذي تحدد الحقول التي يعيدها أعمدة الجدول المؤقت temp_table.

تجدر الإشارة إلى أن هذه الصيغة لن تقوم بملء الجدول المؤقت بالقيم التي يعيدها الاستعلام، ولكن سنحتاج إلى تعبير منفصل لإدراج البيانات ضمن هذا الجدول والذي سيأخذ الصيغة:

```
Insert Into temp_table query;
```

مثال:

إذا أردنا إنشاء الجدول المؤقت myTemp والذي يحوي الحقول productName وproductPrice المستخلصة من الجدول Products يمكننا كتابة الصيغة:

```
CREATE GLOBAL TEMPORARY TABLE myTemp AS
select productName , productPrice from Products;
```

وكما ذكرنا لن يكون للاستعلام السابق أي دور في ملء الجدول المؤقت بالبيانات، لذا سنحتاج إلى الصيغة التالية لملء الجدول:

```
Insert Into myTemp select productName , productPrice from Products;
```

الجداول المؤقتة في Oracle:

تستخدم قواعد بيانات Oracle لإنشاء جدول مؤقت التعبير:
CREATE GLOBAL TEMPORARY TABLE
عوضاً عن CREATE TABLE المستخدم في SQL Server.

الجداول المؤقتة

استخدام الجداول المؤقتة في DB2:

تستخدم قواعد بيانات DB2 التعبير DECLARE GLOBAL TEMPORARY TABLE لإنشاء جدول مؤقت، ويكون لدينا حالتين: إما إنشاء جدول مؤقت وتحديد الحقول التي يتألف منها كما فعلنا في حالة SQL Server أو عبر تعبير Select كما فعلنا في حالة Oracle.

تكمن المشكلة مع DB2 في أننا لا نستطيع إنشاء جدول مؤقت دون إعلام قاعدة البيانات بمكان توضع هذا الجدول، إذ لابد لنا من إنشاء ما يسمى مساحة الجدول. تستخدم الجداول المؤقتة مساحة من النوع userTemporary.

تمكن مساحة الجدول من النوع userTemporary جميع مستخدمي قاعدة البيانات من الوصول إلى هذا الجدول بعكس مساحة الجدول من النوع system التي لا تسمح إلا لنظام إدارة قاعدة البيانات فقط بالوصول إليها. كما تُحذف مساحة الجدول من النوع userTemporary تلقائياً في حال إلغاء الاتصال.

وعليه، تكون مراحل إنشاء جدول مؤقت في DB2 هي التالية:

1- إنشاء مساحة الجدول بالصيغة:

```
CREATE USER TEMPORARY SPACE table_space MANAGED BY SYSTEM USING
(‘path’);
```

2- إنشاء الجدول المؤقت بالصيغة:

```
DECLARE GLOBAL TEMPPORARY TABLE  temp_table
(Field1 Field1Type , Field2 Field2Type) IN table_space;
```

3- يمكننا الآن ملء الجدول باستخدام تعليمة Insert Into

مثال:

لإنشاء الجدول المؤقت الذي يحتوي أسماء وأرقام مستخدمي الهاتف الذين أجروا اتصالات برقم معين، نستخدم الصيغة:

```
CREATE USER TEMPORARY SPACE tempSpace MANAGED BY SYSTEM USING
(‘c:\temp_space’);
```

ثم نقوم بإنشاء الجدول المؤقت وذلك اعتماداً على الصيغة:

```
DECLARE GLOBAL TEMPPORARY TABLE  tempCallers
(Name varchar(50) , Number varchar (15)) IN tempSpace;
```

ثم نقوم بملء الجدول اعتماداً على الصيغة:

```
Insert Into tempCallers
Select Distinct callerName AS Name , callerNumber AS Number from
Callers where Destination = ‘62918763’;
```

الجداول المؤقتة

تستخدم قواعد بيانات DB2 التعبير DECLARE GLOBAL TEMPORARY TABLE لإنشاء جدول مؤقت، ويكون لدينا حالتين: إما إنشاء جدول مؤقت وتحديد الحقول التي يتألف منها كما فعلنا في حالة SQL Server أو عبر تعبير Select كما فعلنا في حالة Oracle.

تكمن المشكلة مع DB2 في أننا لا نستطيع إنشاء جدول مؤقت دون إعلام قاعدة البيانات بمكان توضع هذا الجدول، إذ لابد لنا من إنشاء ما يسمى مساحة الجدول. تستخدم الجداول المؤقتة مساحة من النوع userTemporary.

تمكن مساحة الجدول من النوع userTemporary جميع مستخدمي قاعدة البيانات من الوصول إلى هذا الجدول بعكس مساحة الجدول من النوع system التي لا تسمح إلا لنظام إدارة قاعدة البيانات فقط بالوصول إليها. كما تُحذف مساحة الجدول من النوع userTemporary تلقائياً في حال إلغاء الاتصال.

الجداول المؤقت

استخدام الجداول المؤقتة في mySQL:

تستخدم mySQL التعبير CREATE TEMPORARY TABLE لإنشاء الجداول المؤقتة وهو شبيه بالتعبير CREATE TABLE المستخدم لإنشاء الجداول العادية.

تشكل الجداول المؤقتة في mySQL أحد آليات الالتفاف على عدم دعم mySQL للاستعلامات الفرعية:

```
CREATE TEMPPORARY TABLE temp_table
(Field1 Field1Type , Field2 Field2Type;
```

مثال:

يراد إنشاء الجدول المؤقت myTemp الذي يحتوي أسماء المواد التي تم بيع كمية أكبر من 100 منها ليتم إعادة طلبها من المستودع نستخدم الصيغة:

```
CREATE TEMPORARY TABLE myTemp (Name varchar(50));
```

إذا أردنا ملء هذا الجدول نستخدم الصيغة:

```
Insert Into myTemp select Name , sum (Quantity) from Sales
Group By Name
Having sum (Quantity) > 100;
```

الجداول المؤقتة

تستخدم mySQL التعبير CREATE TEMPORARY TABLE لإنشاء الجداول المؤقتة وهو شبيه بالتعبير CREATE TABLE المستخدم لإنشاء الجداول العادية.

تشكل الجداول المؤقتة في mySQL أحد آليات الالتفاف على عدم دعم mySQL للاستعلامات الفرعية:

استخدام الفهارس

تعتبر الفهارس أحد أغراض قاعدة البيانات التي صُممت لزيادة سرعة العمليات وذلك عبر توليد جداول بحث داخلية. إذ يشبه فهرس قاعدة المعطيات، فهرس كتاب.

يُحسن استخدام الفهارس الأداء في حال كانت الاستعلامات المستخدمة من النوع Select، ويضر بالأداء في حال كانت العمليات هي عمليات حذف وإدراج وتعديل أي Update, Delete وInsert لأنه سيكون مطلوب من تلك العمليات أن تطبق على الجداول الأساسية وعلى الفهارس التابعة للجدول.

يمكننا القول أن استخدام الفهارس يحسن الأداء بشكل يتجاوز ما قد تسببه عمليات التعديل والحذف من تراجع في الأداء.

يمكننا إنشاء فهرس أو مجموعة من الفهارس على جدول معين بحيث يمكن لكل فهرس أن يعمل على حقل أو على مجموعة من الحقول.

عندما تتم فهرسة جدول حسب حقل معين، تتم فهرسة سجلات هذا الجدول حسب قيمة هذا الحقل ونمطه مما يسمح بإيجاد هذه القيمة سريعاً عندما نجري عملية بحث معتمدة على الحقل المفهرس.

مثال:

إذا كان لدينا جدول كبير يحتوي أسماء الزبائن ومعلوماتهم فإن وجود فهرس على حقل اسم الزبون سيسرع عمليات البحث والاستعلام التي تستخدم الاسم. لأن عملية البحث هنا لا تتم بمسح كامل للجدول الأصلي بل بالبحث ضمن جدول الفهرس أولاً قبل الوصول إلى القيم المطابقة في الجدول الأصلي.

انتبه:

يؤثر استخدام عدد كبير من الفهارس سلبياً على الأداء.

استخدام الفهارس

تعتبر الفهارس أحد أغراض قاعدة البيانات التي صُممت لزيادة سرعة العمليات وذلك عبر توليد جداول بحث داخلية. إذ يشبه فهرس قاعدة المعطيات، فهرس كتاب.

يُحسن استخدام الفهارس الأداء في حال كانت الاستعلامات المستخدمة من النوع Select، ويضر بالأداء في حال كانت العمليات هي عمليات حذف وإدراج وتعديل أي Update, Delete وInsert لأنه سيكون مطلوب من تلك العمليات أن تطبق على الجداول الأساسية وعلى الفهارس التابعة للجدول.

يمكننا القول أن استخدام الفهارس يحسن الأداء بشكل يتجاوز ما قد تسببه عمليات التعديل والحذف من تراجع في الأداء.

يمكننا إنشاء فهرس أو مجموعة من الفهارس على جدول معين بحيث يمكن لكل فهرس أن يعمل على حقل أو على مجموعة من الحقول.

عندما تتم فهرسة جدول حسب حقل معين، تتم فهرسة سجلات هذا الجدول حسب قيمة هذا الحقل ونمطه مما يسمح بإيجاد هذه القيمة سريعاً عندما نجري عملية بحث معتمدة على الحقل المفهرس.

إنشاء الفهارس

تطبق الفهارس عادةً على الأعمدة المستخدمة في التعابير Where وOrder By وتلك المستخدمة مع التعبير ON الخاص بالتعبير Join.

كما يفيد استخدام الفهارس مع الحقول التي يقل فيها تكرار القيم حيث تساعد الفهارس على الحصول على اصطفائية عالية.

للفهارس نوعين:

- فهارس فريدة أي لا تسمح بتكرار قيم في الحقل المفهرس.
- فهارس غير فريدة تسمح بتكرار القيم ضمن الحقل المفهرس.

لسنا بحاجة إلى فهرسة المفتاح الرئيسي أو الحقول ذات القيد UNIQUE لأن نظام قاعدة البيانات يولد تلك الفهارس تلقائياً ويقوم بإزالتها حال إزالة صفة المفتاح الرئيسي أو قيد UNIQUE عن هذه الحقول.

لإنشاء فهرس فريد أي لا يسمح بتكرار قيم الحقل المفهرس نستخدم الصيغة:

```
CREATE UNIQUE INDEX index_name ON tableName (FieldName);
```

ولإنشاء فهرس غير فريد أي يسمح بتكرار قيم الحقل المفهرس نستخدم الصيغة:

```
CREATE INDEX index_name ON tableName (FieldName);
```

مثال:

لدينا الجدول Phonebook الذي يحتوي أسماء الأشخاص Name، وأرقام هواتفهم Number وتصنيفهم Category: أصدقاء، عائلة، الخ... فإذا أردنا إنشاء فهرس فريد على حقل فسيكون الاختيار الصائب هو الحقل Number علما أن حقل الاسم هو حقل مفتاح رئيسي مفهرس تلقائياً، وحقل التصنيف Category لا يصلح ليكون فهرس فريد بسبب تكرار قيمه. لذلك ننشئ الفهرس بالصيغة:

```
CREATE UNIQUE INDEX myIndex ON Phonebook (Number);
```

هذه الصيغة ستسرع إعادة نتائج البحث في استعلام من الشكل:

```
Select * from Phonebook where Number = '5437268';
```

انتبه:

في حالة الفهارس على حقول سلاسل المحارف، لا يتم البحث ضمن جدول الفهرس في حال تم الاستعلام عن سلسلة محرفية جزئية (من السلسلة المحرفية الأساسية) لا تبدأ من بداية السلسلة الأصلية ففي استعلام من الشكل:

```
Select * from tableName where strColumn like '%substr';
```

لا يُستخدم الفهرس المنشأ على الحقل strColumn وستتم عملية مسح كاملة للجدول الأساسي لإتمام عملية البحث.

استخدام الفهارس

تطبق الفهارس عادةً على الأعمدة المستخدمة في التعابير Where وOrder By وتلك المستخدمة مع التعبير ON الخاص بالتعبير Join.

كما يفيد استخدام الفهارس مع الحقول التي يقل فيها تكرار القيم حيث تساعد الفهارس على الحصول على اصطفائية عالية.

للفهارس نوعين:

- فهارس فريدة أي لا تسمح بتكرار قيم في الحقل المفهرس.
- فهارس غير فريدة تسمح بتكرار القيم ضمن الحقل المفهرس.

لسنا بحاجة إلى فهرسة المفتاح الرئيسي أو الحقول ذات القيد UNIQUE لأن نظام قاعدة البيانات يولد تلك الفهارس تلقائياً ويقوم بإزالتها حال إزالة صفة المفتاح الرئيسي أو قيد UNIQUE عن هذه الحقول.

حذف الفهارس

لحذف فهرس نستخدم في SQL التعبير DROP INDEX. لكن يحتاج تطبيقه على قواعد البيانات المختلفة، تعديلات طفيفة في الصيغة ففي SQL Server نحتاج إلى تحديد اسم الجدول والفهرس، أي تكون الصيغة بالشكل:

```
DROP INDEX myTable.myIndex;
```

أما في Oracle وDB2 فلا نحتاج إلى اسم الجدول فتصبح الصيغة من الشكل:

```
DROP INDEX myIndex;
```

وفي MySQL نحتاج أيضاً لتحديد اسم الجدول واسم الفهرس كما يلي:

```
DROP INDEX myIndex ON myTable;
```

مثال:

إذا أردنا حذف الفهرس numberIndex من الجدول Phonebook نستخدم الصيغة في SQL Server:

```
DROP INDEX Phonebook.numberIndex;
```

وفي Oracle وDB2 نستخدم الصيغة:

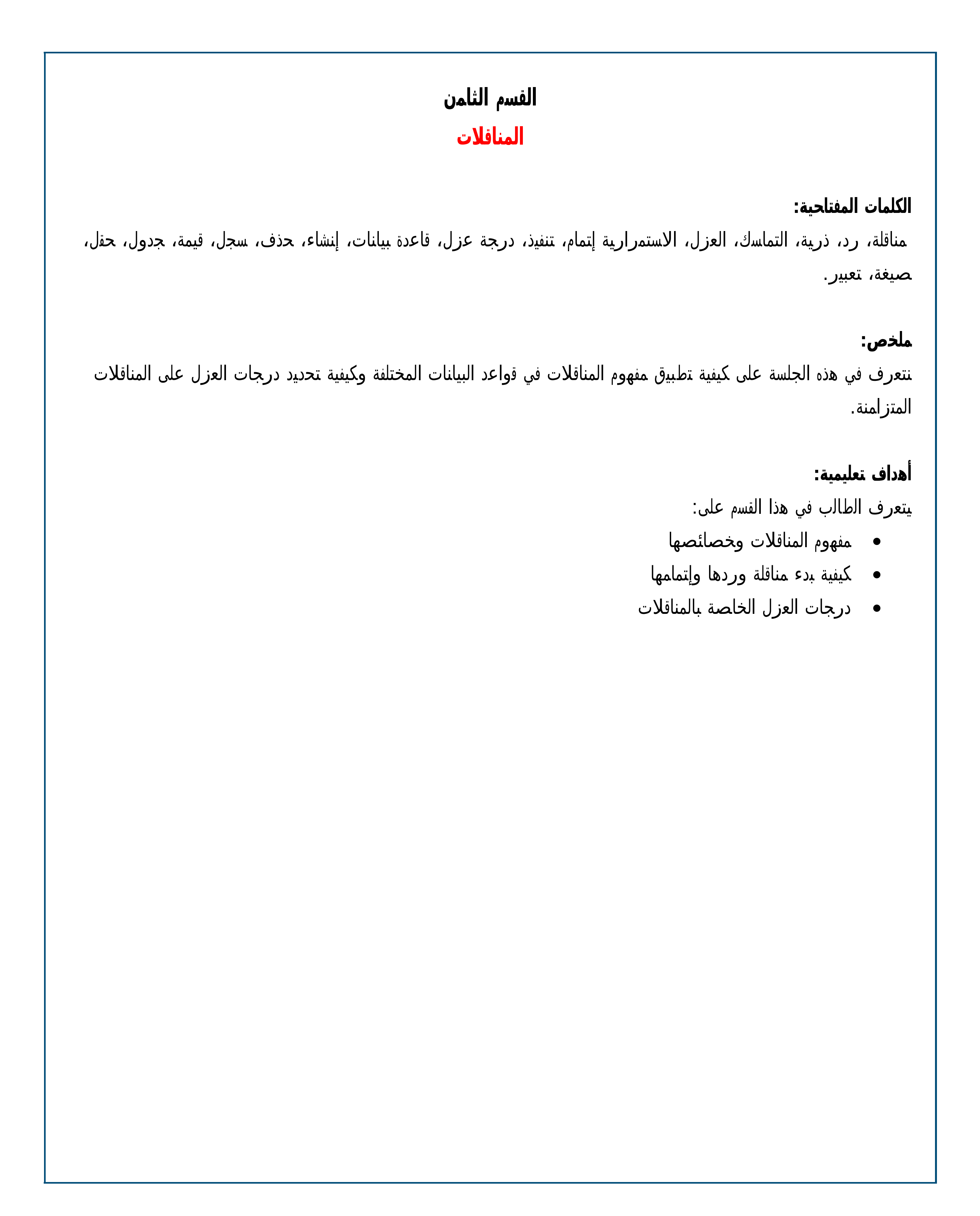
```
DROP INDEX numberIndex;
```

أما في My SQL فنستخدم:

```
DROP INDEX numberIndex ON Phonebook;
```

حذف الفهارس

لحذف فهرس نستخدم في SQL التعبير DROP INDEX. لكن يحتاج تطبيقه على قواعد البيانات المختلفة، تعديلات طفيفة في الصيغة ففي SQL Server وMySQL نحتاج إلى تحديد اسم الجدول والفهرس. أما في Oracle وDB2 فلا نحتاج إلى اسم الجدول.



القسم الثامن

المناقلات

الكلمات المفتاحية:

مناقلة، رد، ذرية، التماسك، العزل، الاستمرارية إتمام، تنفيذ، درجة عزل، قاعدة بيانات، إنشاء، حذف، سجل، قيمة، جدول، حقل، صيغة، تعبير.

ملخص:

نتعرف في هذه الجلسة على كيفية تطبيق مفهوم المناقلات في قواعد البيانات المختلفة وكيفية تحديد درجات العزل على المناقلات المتزامنة.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- مفهوم المناقلات وخصائصها
- كيفية بدء مناقلة وردها وإتمامها
- درجات العزل الخاصة بالمناقلات

المناقلات1

هل سبق لك وفكرت ما الذي سيحدث في حال انقطاع التيار الكهربائي أو حدوث خلل متعلق بالعتاد الصلب أو البرمجيات أثناء عملية تحويل مصرفي الكتروني أو عملية شراء عبر الانترنت. فغالباً ما تتكون هذه العمليات ككل من أكثر من عملية جزئية. هل خطر لك ما الذي سيحدث إذا فشلت إحدى تلك العمليات الجزئية؟

لنأخذ حالة شخص أراد شراء سلعة بسعر 200$ عبر موقع تجارة الكتروني عن طريق بطاقته الائتمانية.
أبسط سيناريو لتلك العملية سيكون سحب المبلغ من حسابه وإيداعه في الحساب الخاص بموقع التجارة الالكتروني وإذا أردنا التعبير عن هذا بصيغة SQL:

```
UPDATE creditAccount set
creditBalance=creditBalance - 200 where creditNumber='345276778543';
```

```
UPDATE SiteAccount set
SiteBalance=SiteBalance  + 200 where accountNumber=345231;
```

الآن لنتخيل أنّ شيئاً ما حدث بعد تنفيذ صيغة SQL الأولى منع تنفيذ الصيغة الثانية، ما سيحدث أن عملية سحب المبلغ من حساب البطاقة الائتمانية قد تم، لكن عملية الإيداع في الحساب الخاص بالموقع لم تتم أو فشلت أي خسر صاحب البطاقة المبلغ دون أن تتم عملية الشراء.

أوجدت أنظمة قواعد البيانات العلائقية حلاً لتلك المشكلة عن طريق ما يسمى بالمناقلات وسنستعرض فيمايلي بالتفصيل ما هي المناقلات ، القوانين التي يجب أن تتبعها المناقلة، كيفية استخدام المناقلات ، ومشاكل الأداء التي قد يطرحها استخدام المناقلات.

المناقلات1

هل سبق لك وفكرت ما الذي سيحدث في حال انقطاع التيار الكهربائي أو حدوث خلل متعلق بالعتاد الصلب أو البرمجيات أثناء عملية تحويل مصرفي الكتروني أو عملية شراء عبر الانترنت. فغالباً ما تتكون هذه العمليات ككل من أكثر من عملية جزئية. هل خطر لك ما الذي سيحدث إذا فشلت إحدى تلك العمليات الجزئية؟

لنأخذ حالة شخص أراد شراء سلعة بسعر 200$ عبر موقع تجارة الكتروني عن طريق بطاقته الائتمانية.
أبسط سيناريو لتلك العملية سيكون سحب المبلغ من حسابه وإيداعه في الحساب الخاص بموقع التجارة الالكتروني حيث يمكن التعبير عن هاتين العمليتين بصيغتي SQL الأولى تنفذ عن عملية السحب، والثانية عملية الدفع.

الآن لنتخيل أنّ شيئاً ما حدث بعد تنفيذ صيغة SQL الأولى منع تنفيذ الصيغة الثانية، ما سيحدث أن عملية سحب المبلغ من حساب البطاقة الائتمانية قد تم، لكن عملية الإيداع في الحساب الخاص بالموقع لم تتم أو فشلت أي خسر صاحب البطاقة المبلغ دون أن تتم عملية الشراء.

أوجدت أنظمة قواعد البيانات العلائقية حلاً لتلك المشكلة عن طريق ما يسمى بالمناقلات وسنستعرض فيمايلي بالتفصيل ما هي المناقلات ، القوانين التي يجب أن تتبعها المناقلة، كيفية استخدام المناقلات ، ومشاكل الأداء التي قد يطرحها استخدام المناقلات.

المناقلات2

ماهي المناقلة:

المناقلة هي وحدة عمل منطقية تتألف من سلسة متتابعة من العمليات المنفصلة التي تحول النظام من وضعية استقرار إلى وضعية استقرار أخرى.

- يجب أن تكون البيانات في أي مرحلة من مراحل العمل في وضع مستقر.
- تعتبر المناقلة ناجحة إذا نجحت جميع العمليات المنفصلة المتسلسلة التي تتألف منها المناقلة.
- تعتبر المناقلة فاشلة إذا فشلت إحدى العمليات المنفصلة المتسلسلة التي تتألف منها المناقلة لأي سبب.
- يتم حفظ جميع التغييرات التي تمت من بدء المناقلة فإذا نجحت المناقلة تم تثبيت هذه التغيرات، أما إذا فشلت نتيجة فشل إحدى العمليات فيتم إعادة النظام إلى وضعه قبل بداية المناقلة والتراجع عن جميع التعديلات التي سببتها العمليات السابقة ضمن نفس المناقلة. تسمى عملية التراجع تلك بردّ المناقلة.
- عندما تصبح التعديلات التي قامت بها المناقلة دائمة نقول أن المناقلة قد تمّت.
- لا يمكن لمناقلة إلا أن تكون ناجحة أو فاشلة أي لا يوجد ما يسمى مناقلة ناجحة جزئياً.

المناقلات بصورة عامة وليس فقط تلك المطبقة في قواعد البيانات يجب أن تحقق مجموعة من الخصائص تسمى خصائص (ACID).

المناقلات3

خصائص المناقلة (ACID):

1- **الذرية:** تعود إلى طبيعة الكل أو اللاشيء الخاصة بالمناقلة. أي إما أن يتم تطبيق المناقلة كاملة أو ألا تتمّ المناقلة أبداً. تعتبر هذه الخاصية الخاصية الرئيسية للمناقلات والأكثر تعبيراً عن طبيعتها.
2- **التماسك:** تعني أن المناقلة يجب أن تنقل النظام من وضع مستقر ومتسق إلى وضع آخر مستقر ومتسق. إذ لا يجب على المناقلة ككل كسر أي قاعدة من قواعد العمل الخاصة بالبيئة التي تطبق فيها. تقع مسؤولية تحقيق هذه الخاصية على المبرمج لكي يحرص على ألا تكسر أي مناقلة ناجحة قواعد العمل المتبعة.
3- **العزل:** تعني أن كل مناقلة يجب أن تعمل مستقلة تماماً عن أية مناقلة أخرى أو عن العمليات الأخرى التي تتم في نفس الوقت. بمعنى آخر يجب أن يبقى أي تعديل تقوم به المناقلة شفافاً تماماً بالنسبة للعمليات التي تحدث خارج المناقلة إلى أن تنجح المناقلة وتصبح التغييرات التي قامت بها دائمة، عندها يمكن لتلك التغيرات أن تخرج إلى العلن وتصبح مرئية.
4- **الاستمرارية:** يجب أن تكون نتائج المناقلة الناجحة قادرة على الصمود ومقاومة الأحداث حتى الأحداث المتعلقة بانهيار النظام ككل.

المناقلات4

تختلف الطريقة التي تعمل فيها المناقلات تبعاً لنظام قواعد البيانات المستخدم ولكن المبادئ والمراحل العامة موحدة لجميع أنواع قواعد البيانات.

تحديد بداية مناقلة:

تحدد بداية مناقلة بحسب المعيار SQL99 بالتعبير START TRANSACTION لكن هذا التعبير غير مستخدم من قبل قواعد بيانات DB2 وOracle وSQL Server وMySQL.

صيغة بدء المناقلة في SQL Server هي:

```
BEGIN TRANSACTION transaction_name;
```

وفي MySQL صيغة بدء المناقلة من الشكل:

```
BEGIN;
```

وفي Oracle يتم بدء المناقلة عند بدء تنفيذ أول استعلام. كذلك الأمر في DB2 إذاً لن نحتاج إلى تعبير بدء المناقلة في قواعد البيانات من هذا النوع.

أما قواعد بيانات Access فلاتدعم المناقلات.

مثال:

إذا أردنا بدء مناقلة باسم Checking تكون الصيغة في SQL Server:

```
BEGIN TRANSACTION Checking;
```

وفي MySQL تبقى الصيغة:

```
BEGIN;
```

المناقلات4

تختلف الطريقة التي تعمل فيها المناقلات تبعاً لنظام قواعد البيانات المستخدم ولكن المبادئ والمراحل العامة موحدة لجميع أنواع قواعد البيانات.

تحديد بداية مناقلة:

تحدد بداية مناقلة بحسب المعيار SQL99 بالتعبير START TRANSACTION لكن هذا التعبير غير مستخدم من قبل قواعد بيانات DB2 وOracle وSQL Server وMySQL.

فإن صيغة بدء المناقلة في SQL Server هي BEGIN TRANSACTION.

وفي MySQL تكون صيغة بدء المناقلة هي BEGIN.

في Oracle يتم بدء المناقلة عند بدء تنفيذ أول استعلام كذلك الأمر في DB2 إذاً لن نحتاج إلى تعبير بدء المناقلة في قواعد البيانات من هذا النوع.

أما قواعد بيانات Access فلاتدعم المناقلات.

المناقلات5

نقاط الحفظ:

تعتبر نقاط الحفظ نقاط مرجعية أو نقاط استرجاع ضمن المناقلة. إذ يمكننا عند استعمال الأمر الخاص برد المناقلة أو التراجع عنها أن نحدد نقطة الحفظ التي يجب أن تعود إليها.

العبارة التي يحددها المعيار SQL99 لإنشاء نقطة حفظ هي العبارة SAVEPOINT وهي من الصيغة:

```
SAVEPOINT savepoint_name;
```

هذه الصيغة مدعومة من DB2 وOracle أما SQL Server فتستخدم العبارة:

```
SAVE TRANSACTION savepoint_name;
```

أما قواعد بيانات MySQL فلا تدعم نقاط الحفظ.

مثال:
لإنشاء نقطة حفظ باسم BeforeChange في قواعد بيانات DB2 وOracle نكتب الصيغة:

```
SAVEPOINT BeforeChange;
```

وفي قواعد بيانات SQL Server تصبح الصيغة:

```
SAVE TRANSACTION BeforeChange;
```

المناقلات5

نقاط الحفظ:
تعتبر نقاط الحفظ نقاط مرجعية أو نقاط استرجاع ضمن المناقلة. إذ يمكننا عند استعمال الأمر الخاص برد المناقلة أو التراجع عنها أن نحدد نقطة الحفظ التي يجب أن تعود إليها.
العبارة التي يحددها المعيار SQL99 لإنشاء نقطة حفظ في قواعد بيانات DB2 وOracle هي العبارة SAVEPOINT.
أما قواعد بيانات SQL Server فتستخدم عبارة SAVE TRANSACTION.
أما قواعد بيانات MySQL فلا تدعم نقاط الحفظ.

المناقلات6

رد أو إلغاء مناقلة:

كما رأينا سابقاً عندما تظهر مشكلة ما أثناء تنفيذ مناقلة نستطيع رد المناقلة أو التراجع عنها إلى نقطة البداية أو إلى نقطة حفظ يتم تحديدها.

صيغة رد مناقلة بحسب المعيار SQL99 هي كالتالي:

```
ROLLBACK [WORK] [TO SAVEPOINT savepoint_name];
```

هذا التعبير صالح في قواعد بيانات Oracle وDB2 ويعمل ولكن بدون تمكين خيار TO SAVEPOINT في قواعد بيانات MySQL لأن MySQL كما علمنا لا تدعم نقاط الحفظ.

أما SQL Server فيستخدم الصيغة:

```
ROLLBACK TRANSACTION [<transaction_name>|<savepoint_name>];
```

مثال:

لدينا المناقلة من الشكل:

```
BEGIN TRANSACTION myTransaction;
Update Accounts SET myBalance = myBalance-100;
SAVE TRANSACTION mySavePoint;
Update Products SET Quantity = Quantity-1;
```

من صيغة المثال نستطيع استنتاج أننا طلبنا من SQL Server استخدام BEGIN TRANSACTION و SAVE TRANSACTION.

في حال أردنا رد المناقلة أي التراجع إلى نقطة الحفظ mySavePoint يمكننا إضافة السطر التالي إلى نهاية الصيغة:

```
ROLLBACK TRANSACTION myTransaction;
```

إذا أردنا كتابة نفس الصيغة في قواعد بيانات Oracle أو DB2 تصبح بالشكل:

```
Update Accounts SET myBalance = myBalance-100;
SAVEPOINT mySavePoint;
Update Products SET Quantity = Quantity-1;
ROLLBACK TO mySavePoint;
```

أما في mySQL والتي لاتدعم نقاط الحفظ فنستخدم التعبير:

```
BEGIN;
Update Accounts SET myBalance = myBalance-100;
Update Products SET Quantity = Quantity-1;
ROLLBACK;
```

المناقلات6

رد أو إلغاء مناقلة:

كما رأينا سابقاً عندما تظهر مشكلة ما أثناء تنفيذ مناقلة نستطيع رد المناقلة أو التراجع عنها إلى نقطة البداية أو إلى نقطة حفظ يتم تحديدها.

المناقلات7

إتمام أو تنفيذ مناقلة:

إذا تم تنفيذ جميع أوامر SQL ضمن مناقلة يمكنك كتابة التعبير COMMIT الذي سيخبر قاعدة البيانات بتأكيد وتثبيت التغييرات التي أحدثتها مناقلة. بعد هذه المرحلة لا يمكن التراجع عن المناقلة باستخدام التعبير ROLLBACK.

الصيغة التي يحددها المعيار SQL99 لإتمام مناقلة هي:

```
COMMIT [WORK];
```

هذه الصيغة مدعومة من قواعد بيانات mySQL, SQL Server, Oracle, DB2.

كما يوفر SQL Server في الصيغة البديلة COMMIT TRANSACTION إمكانية لإدخال معامل يحدد اسم المناقلة المراد إتمامها وتكون الصيغة من الشكل:

```
COMMIT TRANSACTION transaction_name;
```

مثال:

سنعود إلى مثالنا الذي بدأنا به جلستنا وسنستخدم معه التعابير الخاصة بالمناقلة لبدء المناقلة وردها وإتمامها:

```
BEGIN TRANSACTION
SAVE TRANSACTION beforeChange
UPDATE creditAccount set
creditBalance=creditBalance - 200 where creditNumber='345276778543';
ROLLBACK TRANSACTION beforeChange
UPDATE SiteAccount set
SiteBalance=SiteBalance  + 200 where accountNumber=345231;
COMMIT TRANSACTION
```

المناقلات7

إتمام أو تنفيذ مناقلة:

إذا تم تنفيذ جميع أوامر SQL ضمن مناقلة يمكنك كتابة التعبير COMMIT الذي سيخبر قاعدة البيانات بتأكيد التغييرات التي أحدثتها مناقلة وتثبيتها. بعد هذه المرحلة لا يمكن التراجع عن المناقلة باستخدام التعبير ROLLBACK.

المناقلات8

إتمام أو تنفيذ مناقلة آلياً:

تدعم بعض قواعد البيانات إتمام أو تنفيذ مناقلة آلياً مثل قواعد بيانات SQL Server وmySQL وتعتبر هذه الوضعية هي الوضعية التلقائية في قواعد البيانات تلك.
الإتمام أو التنفيذ الآلي للمناقلة يعني أن قاعدة البيانات ستعامل كل استعلام على أساس كونه مناقلة منفصلة حتى بدون استخدام أوامر بداية أوإتمام أو رد المناقلة، وبعد تنفيذ كل استعلام من هذه الاستعلامات سوف يتم إتمام المناقلة تلقائياً ولايمكن ردها بعد ذلك.

مثال:

إذا كان لدينا الصيغة:

```
Update myTable SET ID = 10;
Update Customers SET customerName = 'Adel';
```

وكان خيار الإتمام الآلي لمناقلات قاعدة البيانات مفعّلاً، سيتم اعتبار كل استعلام من هذين الاستعلامين كمناقلة منفصلة وسيتم إتمامها حال تنفيذها.

لجعل قاعدة البيانات تعتمد هذين الاستعلامين كمناقلة واحدة لابد من صياغة التعبير بالشكل:

```
BEGIN WORK
Update myTable SET ID = 10;
Update Customers SET customerName = 'Adel';
COMMIT WORK;
```

لتفعيل أو تعطيل خيار الإتمام الآلي للمناقلات يمكننا استخدام التعبير التالي في Oracle:

```
SET AUTOCOMMIT ON|OFF;
```

أما في SQL Server نستخدم التعبير:

```
SET IMPLICIT_TRANSACTIONS ON|OFF;
```

أما في DB2 فيتم اختيار هذا الخيار من نافذة Command Center >Options

المناقلات8

إتمام أو تنفيذ مناقلة آلياً:

تدعم بعض قواعد البيانات إتمام أو تنفيذ مناقلة آلياً مثل قواعد بيانات SQL Server وmySQL وتعتبر هذه الوضعية هي الوضعية التلقائية في قواعد البيانات تلك.
الإتمام أو التنفيذ الآلي للمناقلة يعني أن قاعدة البيانات ستعامل كل استعلام على أساس كونه مناقلة منفصلة حتى بدون استخدام أوامر بداية أوإتمام أو رد المناقلة، وبعد تنفيذ كل استعلام من هذه الاستعلامات سوف يتم إتمام المناقلة تلقائياً ولايمكن ردها بعد ذلك.

المناقلات9

اختبار الخطأ في المناقلة:

لم نتكلم حتى الآن عن كيفية اختبار ظهور خطأ ما أو حصول خلل في مناقلة. يساعد هذا الاختبار في اتخاذ قرار حول إتمام المناقلة أو ردها. لن ندخل في تفاصيل إدارة الأخطاء ومعالجتها لأننا سنفرد بحث خاص لمعالجة هذا الموضوع، ولكن لتصبح فكرة المناقلة واستخدامها كاملة لابد لنا من استخدام تقنية اختبار الخطأ هنا.

مثال:

إذا كان لدينا المناقلة من الشكل:

```
BEGIN TRANSACTION myTransaction
Insert Into Graduated (ID, Name) Values (20, 'Samer')
IF @@ERROR <> 0 ROLLBACK TRANSACTION myTransaction
Update Students SET Status = 'Graduated' where ID = 20;
COMMIT TRANSACTION myTransaction
```

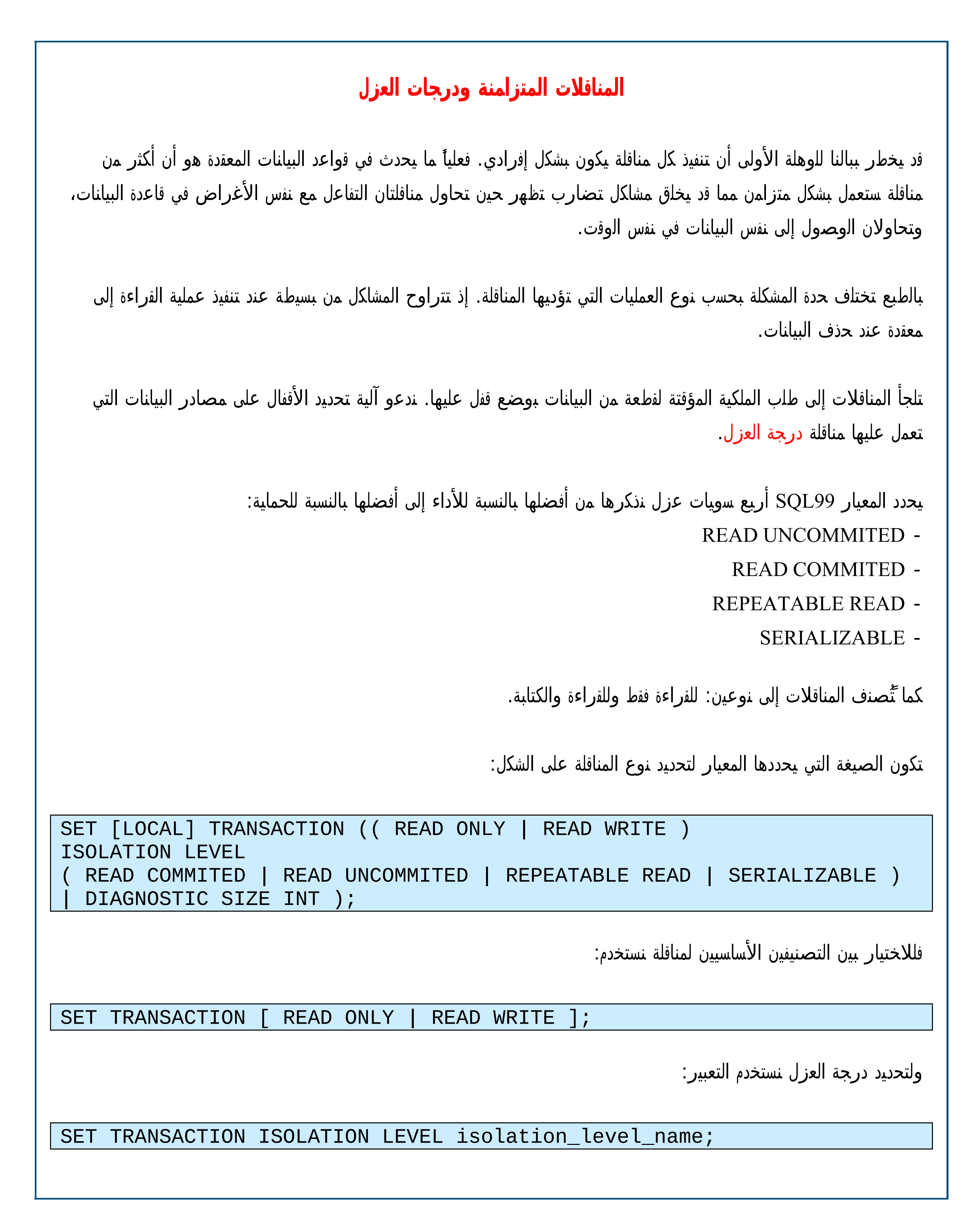
كما نلاحظ هنا:

- بدأنا مناقلة باسم myTransaction
- وحاولنا إضافة سجل المتخرج 'Samer' ذو الرقم 20 إلى جدول المتخرجين،
- ثم اختبرنا ظهور أي خطأ،
  - o فإذا كان هناك خطأ ما سوف يتم رد المناقلة والتراجع عن الاستعلام بالتعبير ROLLBACK،
  - o وإلا فسيتم تنفيذ الاستعلام الثاني الذي سيقوم بتعديل وضع الطالب 'Samer' في جدول الطلاب،
- ثم سيتم إتمام المناقلة بالتعبير COMMIT TRANSACTION.

المناقلات 9

اختبار الخطأ في المناقلة:

لم نتكلم حتى الآن عن كيفية اختبار ظهور خطأ ما أو حصول خلل في مناقلة. يساعد هذا الاختبار في اتخاذ قرار حول إتمام المناقلة أو ردها. لن ندخل في تفاصيل إدارة الأخطاء ومعالجتها لأننا سنفرد بحث خاص لمعالجة هذا الموضوع، ولكن لتصبح فكرة المناقلة واستخدامها كاملة لابد لنا من استخدام تقنية اختبار الخطأ هنا.



المناقلات المتزامنة ودرجات العزل

قد يخطر ببالنا للوهلة الأولى أن تنفيذ كل مناقلة يكون بشكل إفرادي. فعلياً ما يحدث في قواعد البيانات المعقدة هو أن أكثر من مناقلة ستعمل بشكل متزامن مما قد يخلق مشاكل تضارب تظهر حين تحاول مناقلتان التفاعل مع نفس الأغراض في قاعدة البيانات، وتحاولان الوصول إلى نفس البيانات في نفس الوقت.

بالطبع تختلف حدة المشكلة بحسب نوع العمليات التي تؤديها المناقلة. إذ تتراوح المشاكل من بسيطة عند تنفيذ عملية القراءة إلى معقدة عند حذف البيانات.

تلجأ المناقلات إلى طلب الملكية المؤقتة لقطعة من البيانات بوضع قفل عليها. ندعو آلية تحديد الأقفال على مصادر البيانات التي تعمل عليها مناقلة درجة العزل.

يحدد المعيار SQL99 أربع سويات عزل نذكرها من أفضلها بالنسبة للأداء إلى أفضلها بالنسبة للحماية:

- READ UNCOMMITED
- READ COMMITED
- REPEATABLE READ
- SERIALIZABLE

كما تُصنف المناقلات إلى نوعين: للقراءة فقط وللقراءة والكتابة.

تكون الصيغة التي يحددها المعيار لتحديد نوع المناقلة على الشكل:

```
SET [LOCAL] TRANSACTION (( READ ONLY | READ WRITE )
ISOLATION LEVEL
( READ COMMITED | READ UNCOMMITED | REPEATABLE READ | SERIALIZABLE )
| DIAGNOSTIC SIZE INT );
```

فللاختيار بين التصنيفين الأساسيين لمناقلة نستخدم:

```
SET TRANSACTION [ READ ONLY | READ WRITE ];
```

ولتحديد درجة العزل نستخدم التعبير:

```
SET TRANSACTION ISOLATION LEVEL isolation_level_name;
```

انتبه:

درجة العزل التلقائية في SQL Server, DB2, mySQL, Oracle هي READ COMMITED.

المناقلات المتزامنة ودرجات العزل

قد يخطر ببالنا للوهلة الأولى أن تنفيذ كل مناقلة يكون بشكل إفرادي. فعلياً ما يحدث في قواعد البيانات المعقدة هو أن أكثر من مناقلة ستعمل بشكل متزامن مما قد يخلق مشاكل تضارب تظهر حين تحاول مناقلتان التفاعل مع نفس الأغراض في قاعدة البيانات، وتحاولان الوصول إلى نفس البيانات في نفس الوقت.

بالطبع تختلف حدة المشكلة بحسب نوع العمليات التي تؤديها المناقلة. إذ تتراوح المشاكل من بسيطة عند تنفيذ عملية القراءة إلى معقدة عند حذف البيانات.

تلجأ المناقلات إلى طلب الملكية المؤقتة لقطعة من البيانات بوضع قفل عليها. ندعو آلية تحديد الأقفال على مصادر البيانات التي تعمل عليها مناقلة درجة العزل.

يحدد المعيار SQL99 أربع سويات عزل نذكرها من أفضلها بالنسبة للأداء إلى أفضلها بالنسبة للحماية:

- READ UNCOMMITED
- READ COMMITED
- REPEATABLE READ
- SERIALIZABLE

كما تٌصنف المناقلات إلى نوعين: للقراءة فقط وللقراءة والكتابة.

درجات العزل

درجة العزل READ UNCOMMITED:

هي أسرع وأكثر درجات العزل خطورة. تستخدم الصيغة:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITED;
```

عندما تكون المناقلة من هذا النوع لا تكون خاصية العزل المميزة للمناقلات مطبقة وتستطيع المناقلات قراءة التعديلات التي لم يتم إتمامها.

وتظهر مشاكل القراءات الخاطئة حيث قد تقرأ مناقلة مجموعة من البيانات تم تعديلها من قبل مناقلة أخرى غير متممة بـ COMMIT. عندها تكون البيانات التي قامت المناقلة الأولى بقراءتها خاطئة.

يمكننا هنا طرح السؤال التالي: مادامت درجة العزل هذه خطرة وغير كافية لماذا تم تبنيها كحدى درجات العزل؟ والإجابة على هذا السؤال تكمن في أن الحالة التي قد نفكر فيها باستخدام درجة العزل تلك هي حالة توليد تقارير غير حساسة من قاعدة بيانات حيث عمليات الإدراج والتعديل والحذف في حدها الأدنى. مما يساعد على تطبيق العزل مع الحفاظ على مستوى أداء عالي وسرعة جيدة.

درجات العزل

درجة العزل READ COMMITED:

تشكل هذه الدرجة الوضعية التلقائية للمناقلات في أغلب قواعد البيانات.

عند استخدام درجة العزل هذه يتم إقفال جميع الموارد التي تقوم المناقلة بتعديلها إلى أن تتم المناقلة، مما يعني أن جميع عمليات التعديل ستكون محجوبة عن المناقلات الأخرى لحين إتمام المناقلة. إلا أن هذه الدرجة لا تقوم بإقفال السجلات التي تقوم المناقلة بقراءتها.

لتنفيذ هذه الدرجة من العزل نستخدم الصيغة:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITED;
```

لا نستطيع هنا أن نقول أننا أوجدنا الحل الكامل لأن درجة العزل هذه قد تسبب ما يسمى بمشكلة القراءة المتكررة. لتوضيح هذه الفكرة، علينا أن نتخيل أن مناقلة تقوم بقراءة قيمة معينة من قاعدة البيانات، وبما أن العملية هي عملية قراءة فلن يتم إغلاق السجل من قبل هذه المناقلة، لذا يمكن لمناقلة أخرى أن تعدله بينما تكون مناقلتنا الأولى تقوم بعمل آخر، فإذا عادت نفس المناقلة الأولى لتقرأ هذا السجل مرة أخرى ستقرأ فيه قيمة مختلفة.
إذا كان هذا الأمر مقبولاً بالنسبة لك أو إذا أمكنك إيجاد حل لتفادي هذا السيناريو كأن تخزن القيمة التي تحصل عليها في المرة الأولى في متغير مؤقت لتتفادى قراءتها في نفس المناقلة مجدداً عندئذ يمكنك الاستمرار باستخدام درجة العزل READ COMMITED. وإلا ستكون مضطراً إلى الانتقال إلى سوية العزل REPEATABLE READ.

درجات العزل

درجة العزل REPEATABLE READ:

توفر درجة العزل هذه حلاً لمشاكل القراءات الخاطئة والقراءة المتكررة، وذلك بإقفال السجلات التي تقرأها المناقلة وليس تلك التي تعدلها أو تحذفها فقط كما هي الحال مع درجة العزل READ COMMITED.

في هذه الحالة تستطيع أن تضمن أن مناقلة ما ستحصل على نفس القيمة عن قراءة سجل ما لأكثر من مرة في نفس المناقلة.
لايمنع نوع القفل الذي تطبقه هذه الدرجة من العزل، المناقلات الأخرى من قراءة السجل بل يمنعها من تعديله.

طبعاً سندفع ثمن هذا الحل من سرعة الأداء بسبب زمن الانتظار للمناقلات التي تنتظر إلغاء الأقفال عن سجلات معينة لتقوم بتعديلها.

نستخدم لتفعيل درجة العزل هذه الصيغة:

```
SET TRANSACTION ISOLATION LEVEL REPATABLE READ;
```

من جديد نتساءل هل غطينا كل الثغرات ؟ الإجابة هي لا. ما نزال معرضين لمشكلة تسمى بمشكلة الأشباح.

قد يبدو اسم هذه المشكلة غريباً لذلك لفهم التفاصيل لنتخيل السيناريو التالي:

- لدينا مناقلة أولى قامت أحد عملياتها بقراءة جميع السجلات التي تحقق شرط where محدد،
- بنفس الوقت، قامت مناقلة أخرى بإدراج سجل. بهذه العملية لم تقم بكسر أي قاعدة لأنّ عملية إدراج سجل ستقحم سجل جديد ولا تعدل سجل موجود قامت المناقلة الأولى بإقفاله.
- الآن ماذا لو حاولت المناقلة الأولى إعادة قراءة السجلات؟ ستحصل على سجلات إضافية ظهرت كالأشباح التي لم تكن موجودة في الاستعلام السابق ضمن نفس المناقلة.

باختصار تعتبر مشكلة الأشباح مكافئة لمشكلة القراءة المتكررة ولكن بفرق أن المشكلة هنا في السجلات الجديدة التي تم إدراجها.

درجات العزل

درجة العزل SERIALIZABLE:

نستطيع في درجة العزل هذه ضمان عدم قدرة أي مناقلة من إضافة أو تعديل أو حذف أي سجل لأي بيانات تدخل في شرط where لعملية في مناقلة أخرى.

سنعاني بالطبع المزيد من الوقت الضائع وانخفاض الأداء.

الصيغة المستخدمة لتفعيل درجة العزل هذه هي:

```
SET TRANSACTION ISOLATION LEVEL SERIALISABLE;
```

درجات العزل في قواعد البيانات المختلفة

درجات العزل في SQL server:

يدعم SQLserver درجات العزل الأربعة ولكنه لا يدعم المناقلات التي تسمح بالقراءة فقط.

درجة العزل التلقائية في SLQ Server هي READ COMMITED.

الصيغة المستخدمة هي التالية:

```
SET TRANSACTION ISOLATION LEVEL
{ READ COMMITED
| READ UNCOMMITED
| REPEATABLE READ
| SERIALIZABLE
}
```

درجات العزل في Oracle:

يدعم Oracle المناقلات التي تسمح بالقراءة فقط ولكنه لا يدعم درجات العزل READ UNCOMMITED وREPEATABLE READ.

درجة العزل التلقائية في Oracle هي READ COMMITED. كما يؤثر تعبير SET TRANSACTION في Oracle على المناقلة الحالية فقط وليس على المناقلات المنشأة باتصالات أخرى أو أي مناقلات أخرى.

الصيغة المستخدمة هي:

```
SET TRANSACTION
{ {READ ONLY | READ WRITE}
| ISOLATION LEVEL
{ READ COMMITED
| SERIALIZABLE
}};
```

لتغيير درجة العزل لجميع المناقلات الخاصة بجميع جلسات العمل يُستخدم التعبير ALTER SESSION بالصيغة:

```
ALTER SESSION SET ISOLATION_LEVEL SERIALIZABLE;
```

درجات العزل في قواعد البيانات المختلفة

درجات العزل في SQL server:

يدعم SQLserver الدرجات الأربعة للعزل ولكنه لا يدعم المناقلات التي تسمح بالقراءة فقط.

درجة العزل التلقائية في SLQ Server هي READ COMMITED.

درجات العزل في Oracle:

يدعم Oracle المناقلات التي تسمح بالقراءة فقط ولكنه لا يدعم درجات العزل READ UNCOMMITED وREPEATABLE READ.

درجة العزل التلقائية في Oracle هي READ COMMITED. كما يؤثر تعبير SET TRANSACTION في Oracle على المناقلة الحالية فقط وليس على المناقلات المنشأة باتصالات أخرى أو أي مناقلات أخرى.

درجات العزل في قواعد البيانات المختلفة

درجات العزل في MySQL:

يدعم MySQL درجات العزل للمناقلات الخاصة بجداول InnoDB ويستخدم الصيغة:

```
SET [GLOBAL | SESSION ] TRANSACTION ISOLATION LEVEL
{ READ COMMITED
| READ UNCOMMITED
| REPEATABLE READ
| SERIALIZABLE
};
```

يؤثر تعبير SET TRANSACTION فقط على المناقلات التي لم تبدأ بعد ولايؤثر على المناقلات الجارية.
يطبق خيار SESSION التغيير على المناقلات الخاصة بالجلسة الحالية، أما خيار GLOBAL فيجعل التعبير يطبق على جميع المناقلات التي تنشأ لاحقاً في أي جلسة.

درجات العزل في DB2:

تدعم DB2 درجات العزل للمناقلات ولكن تختلف طريقة تطبيقها بعض الشيء مقارنةً مع قواعد البيانات الأخرى فهي تستخدم مع التعبير WITH في صيغة الاستعلام نفسه.

وتكون الصيغة من الشكل:

```
Any query WITH isolation_level;
```

يمكن أن تأخذ isolation_level القيم:

- RR: تعبر عن درجة عزل REPEATABLE READ
- RS: مشابهة لـ REPEATABLE READ
- CS: مقابلة لدرجة عزل READ COMMITED
- UR: مقابلة لدرجة عزل READ UNCOMMITED

مثلاً يمكننا كتابة الصيغة:

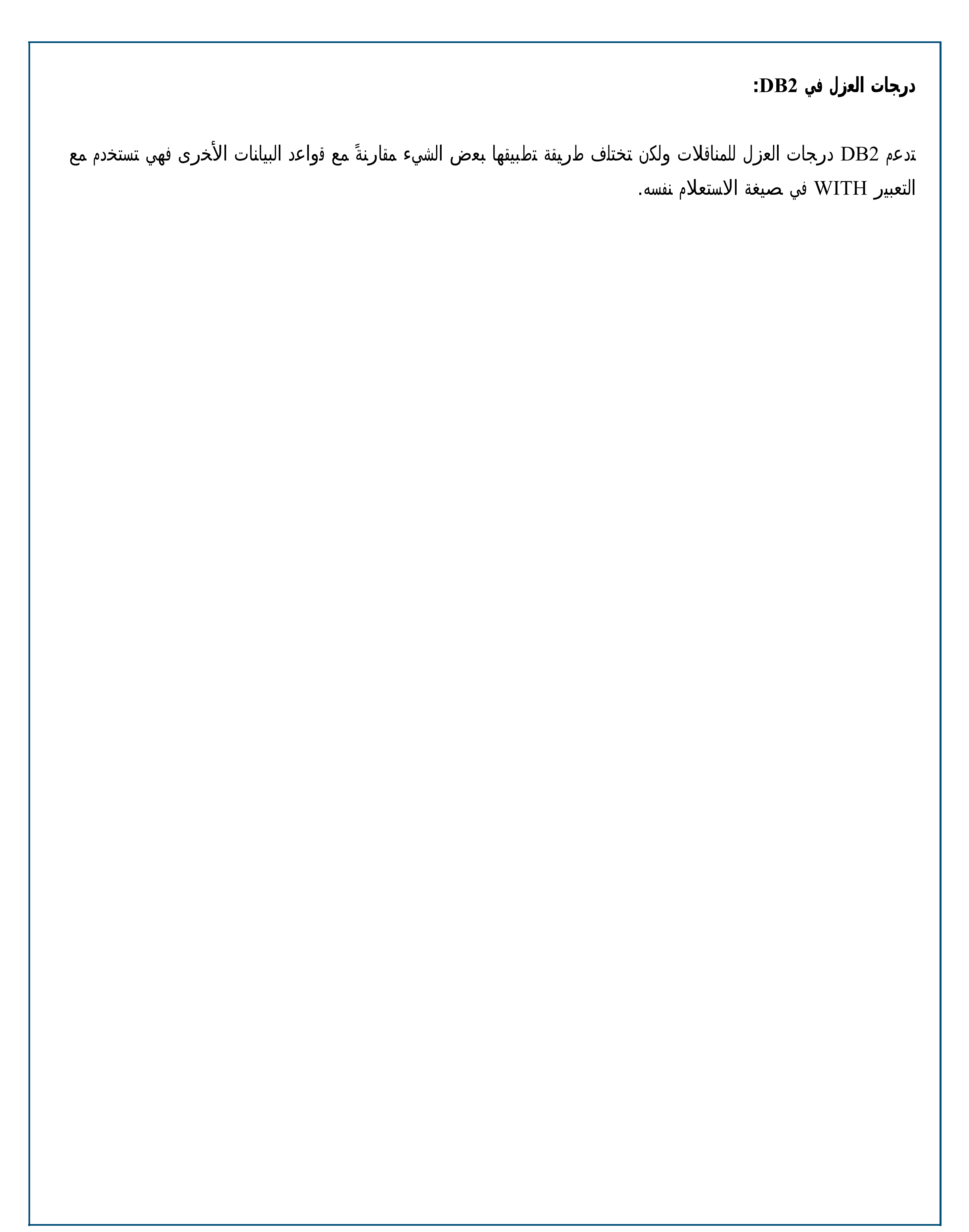
```
Update myTable SET myColumn = 10 Where otherColumn = 5
WITH RR;
```

درجات العزل في قواعد البيانات المختلفة

درجات العزل في MySQL:

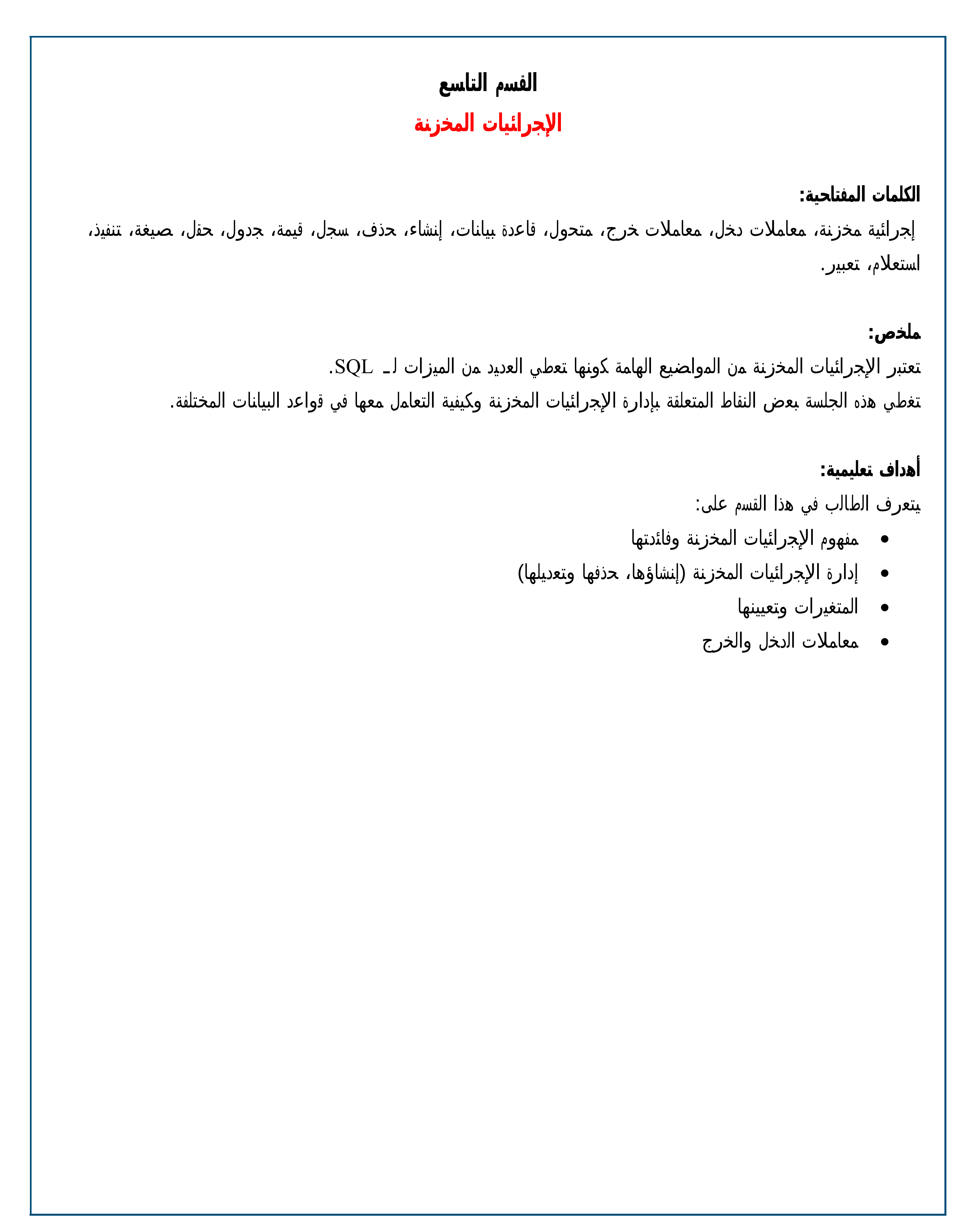
يدعم MySQL درجات العزل للمناقلات الخاصة بجداول InnoDB

في MySQL تعبير SET TRANSACTION يؤثر فقط على المناقلات التي لم تبدأ بعد ولايؤثر على المناقلات الجارية.
خيار SESSION يطبق التغيير على المناقلات للجلسة الحالية أما خيار GLOBAL فيجعل التعبير يطبق على جميع المناقلات التي تنشأ لاحقاً في أي جلسة.



درجات العزل في DB2:

تدعم DB2 درجات العزل للمناقلات ولكن تختلف طريقة تطبيقها بعض الشيء مقارنةً مع قواعد البيانات الأخرى فهي تستخدم مع التعبير WITH في صيغة الاستعلام نفسه.



القسم التاسع

الإجرائيات المخزنة

الكلمات المفتاحية:

إجرائية مخزنة، معاملات دخل، معاملات خرج، متحول، قاعدة بيانات، إنشاء، حذف، سجل، قيمة، جدول، حقل، صيغة، تنفيذ، استعلام، تعبير.

ملخص:

تعتبر الإجرائيات المخزنة من المواضيع الهامة كونها تعطي العديد من الميزات لـ SQL.
تغطي هذه الجلسة بعض النقاط المتعلقة بإدارة الإجرائيات المخزنة وكيفية التعامل معها في قواعد البيانات المختلفة.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- مفهوم الإجرائيات المخزنة وفائدتها
- إدارة الإجرائيات المخزنة (إنشاؤها، حذفها وتعديلها)
- المتغيرات وتعيينها
- معاملات الدخل والخرج

الإجرائيات المُخزَنة 1

تكلمنا فيما سبق عن المناظير وذكرنا أنها تسهل تمثيل الاستعلامات المعقدة دون الحاجة إلى إعادة كتابة هذه الاستعلامات.

الإجرائيات المخزنة عبارة عن آلية تلعب دور مشابه للدور الذي تلعبه المناظير ولكن بصورة موسعة. إذ لا تنحصر قدرة الإجرائيات المخزنة على استعلام واحد بل يمكن لإجرائية مخزنة واحدة أن تقوم بمجموعة من الأعمال كإضافة سجل، ثم تعديل البيانات في سجل آخر، والقيام بالحسابات على مجموعة أخرى من السجلات وذلك اعتباراً من أمر واحد من المستخدم.

عند استخدام الإجرائيات المخزنة نقوم بإنشاء إجرائية مع معامِلات دخل وخرج بحيث يمكننا استدعاؤها بالصورة نفسها المستخدمة في تعبيرات SQL المعيارية.

تحتوي الإجراءات المخزنة تعبير أو مجموعة من تعبيرات SQL والتي تنفذ كجزء من تنفيذ الإجرائية مع إمكانيات التحكم بتدفق التنفيذ أو إضافة حلقات أو التنفيذ الشرطي، مما يعطي الإجرائيات المخزنة تميزاً كبيراً.

تختلف الطريقة التي تطبق فيها قواعد البيانات الإجرائيات المخزنة بالرغم من أن الفكرة الأساسية موحدة.

الإجرائيات المُخزَنة 2

يتم استدعاء الإجرائيات المخزنة باستخدام تعبيرات تعتمد على قاعدة البيانات المستخدمة فهي تستخدم تعبير CALL في Oracle وDB2، وتستخدم التعبير EXECUTE أو EXEC في SQL Server وAccess.
كما تدعم Oracle العبارة EXECUTE أيضاً.

مثال:
إذا أردنا استدعاء إجرائية مخزنة باسم UpdateRec مع تحديد رقم 3 كمعامِل دخل تصبح صيغة الاستدعاء في Oracle وDB2 من الشكل:

```
CALL UpdateRec (3);
```

أما في SQL وAccess فتصبح الصيغة:

```
EXECUTE UpdateRec 3;
```

نلاحظ أننا لم نستخدم الأقواس مع معاملات الدخل والخرج في الصيغة الخاصة بـ SQL Server وAccess.
أما إذ استخدمنا أكثر من مُعامل، فيتم القسم فيما بينها بفاصلة.

الإجرائيات المخزنة 2

يتم استدعاء الإجرائيات المخزنة باستخدام تعبيرات تعتمد على قاعدة البيانات المستخدمة فهي تستخدم تعبير CALL في Oracle وDB2، وتستخدم التعبير EXECUTE أو EXEC في SQL Server وAccess.
كما تدعم Oracle العبارة EXECUTE أيضاً.

إنشاء إجرائية مخزنة

يمكننا إنشاء إجرائية مخزنة باستخدام التعبير CREATE PROCEDURE وذلك حسب الصيغة التالية:

```
CREATE PROCEDURE sp_name (parameter_list)
AS sp_body;
```

حيث sp_name هو اسم الإجرائية المخزنة وParameter_list هو قائمة بمعامِلات الدخل والخرج لهذه الإجرائية مفصولة بفواصل وsp_body هو محتوى أو تعليمات الإجرائية المخزنة.

الصيغة السابقة مدعومة من قواعد بيانات Oracle, DB2, SQL Server وAccess.

تدعم قواعد البيانات السابقة تعبيراتها الخاصة لإنشاء إجرائيات مخزنة ولكننا سنعتمد هذه الصيغة كونها عامة.

مثال:
إذا أردنا إنشاء إجرائية مخزنة لتقوم بحذف سجل من جدول Products اعتماداً على رقم السجل الذي نمرره لتلك الإجرائية، كمعامل دخل، والذي يمثل قيمة الحقل ID لهذا السجل، نستخدم الصيغة:

```
CREATE PROCEDURE deleteProduct (@ProductID INT)
AS
BEGIN
Delete from Products where ID = @ProductID;
END;
```

ما نلاحظه في هذه الصيغة أنه قد تم احتواء جسم الإجرائية المخزنة في التركيب END, BEGIN وتم استخدام معامِل الدخل ProductID كجزء من الاستعلام.

استخدمنا الإشارة @ قبل اسم المعامل لأن الصيغة السابقة هي صيغة الإجرائية في SQL Server.

الآن إذا أردنا استخدام الإجرائية المخزنة التي قمنا بكتابتها، يمكن ببساطة استدعاءها وتنفيذها لحذف السجل ذو قيمة الحقل ID المساوية لـ 8 باستخدام الصيغة:

```
EXECUTE deleteProduct (8);
```

إنشاء إجرائية مخزنة

يمكننا إنشاء إجرائية مخزنة باستخدام التعبير CREATE PROCEDURE.

إنشاء إجرائية مخزنة

إنشاء إجرائية مخزنة في Oracle:

يشبها التعبير الخاص بإنشاء إجرائية مخزنة في Oracle لذاك المستخدم في SQL Server والمطابق للمعيار ANSI وهو من الشكل:

```
CREATE [OR REPLACE] PROCEDURE sp_name (parameter_list)
AS
BEGIN
      Sp_body;
END;
```

نلاحظ في الصيغة السابقة وجود الخيار OR REPLACE الذي يحدد عند إضافته إمكانية إعادة تعريف إجرائية مخزنة في حال وجودها مسبقاً.

أحد الاختلافات الهامة أيضاً هي عدم الحاجة إلى استباق أسماء المعامِلات بالإشارة @.
كذلك ضرورة تعريف المعاملات على أنها معاملات دخل أو خرج أو دخل/ خرج وهذا ما لم نحدده في الصيغة المستخدمة مع قواعد بيانات SQL Server.

انتبه:

يقوم نظام Oracle بإنشاء الإجرائية المخزنة حتى لو كانت تحتوي أخطاء ويمكن إعادة تعريفها بصورة صحيحة أو تعديلها باستخدام الخيار OR REPLACE كما ذكرنا.

مثال:

إذا أردنا إنشاء إجرائية مخزنة تقوم بتعديل قيمة الحقل Status من الجدول Students بحيث يسند معامل الإجرائية لهذا الحقل في حال كانت علامة الطالب أقل من علامة دنيا نستخدم الصيغة:

```
CREATE OR REPLACE PROCEDURE updateStatus
(minMark IN INT, myStatus IN varchar)
AS
BEGIN
Update Students SET Status = myStatus where Mark < minMark;
END;
```

إنشاء إجرائية مخزنة

إنشاء إجرائية مخزنة في Oracle:

يشبها التعبير الخاص بإنشاء إجرائية مخزنة في Oracle لذاك المستخدم في SQL Server والمطابق للمعيار ANSI.

إنشاء إجرائية مخزنة

إنشاء إجرائية مخزنة في mySQL:

لم تكن قواعد بيانات mySQL تدعم الإجرائيات المخزنة في النسخ قبل 5.0.3 ولكن ابتداءً من تلك النسخة أُدخل دعم الإجرائيات المخزنة إلى mySQL.

```
CREATE PROCEDURE sp_name (parameter_list)
BEGIN
      query_body;
END;
```

مثال:

إذا أردنا كتابة إجرائية مخزنة تقوم بإعادة عدد سجلات جدول myTable وتضعه في معامل خرج باسم myCount نستخدم الصيغة:

```
CREATE PROCEDURE getCount (OUT myCount INT)
BEGIN
      SELECT Count (*) Into myCount form myTable;
END;
```

إنشاء إجرائية مخزنة

إنشاء إجرائية مخزنة في mySQL:

لم تكن قواعد بيانات mySQL تدعم الإجرائيات المخزنة في النسخ قبل 5.0.3 ولكن ابتداءً من تلك النسخة أُدخل دعم الإجرائيات المخزنة إلى mySQL.

إنشاء إجرائية مخزنة

إنشاء إجرائية مخزنة في DB2:

تدعم قواعد بيانات DB2 الإجرائيات المخزنة ولكنك لا تستطيع إنشاء إجرائية مخزنة مباشرة من خلال تعليمة SQL بل تحتاج إلى إنشاء وترجمة نَص الإجرائية بصورة منفصلة، وتحتاج أيضاً إلى تثبيت Microsoft Visual C++ لإتمام عملية الترجمة. لن نخوض في تفاصيل هذه العملية ولكن سننوه فقط إلى الصيغة المستخدمة في DB2 والخاصة بالإجرائيات المخزنة.

لا تختلف الصيغة العامة كثيراً عن ما استخدمناه في قواعد البيانات الأخرى وهي من الشكل:

```
CREATE PROCEDURE sp_name (parameter_list)
block_name: BEGIN
sp_body;
END block_name;
```

مثال:

إذا أردنا كتابة إجرائية مخزنة لتقوم بإدراج اسم طالب جديد ورقمه في الجدول Students نكتب نص الإجرائية المخزنة بالشكل:

```
CREATE PROCEDURE
insertStudent (mystudentID INT, mystudentName varchar(50))
Label1: BEGIN
Insert Into Students (studentID, studentName)
Values(myStudentID, mystudentName)
END Label1;
```

إنشاء إجرائية مخزنة

إنشاء إجرائية مخزنة في Ms Access:

لا يدعم Access الإجرائيات المخزنة ولكنه يسمح لك بإنشاء استعلامات مخزّنة باستخدام التعبير CREATE PROCEDURE. كما يُمكنك استعمال اسم الاستعلام المخزن الذي أنشأته (كما هي الحال عند استعمال أسماء المناظير ضمن استعلام). ففي حال كان لدينا الاستعلام المخزن باسم getProducts تستطيع كتابة التعبير:

```
Select productName from getProducts;
```

كما تستطيع استدعاء الاستعلام المخزن بنفس الأسلوب الذي تستدعى فيه الإجراءات المخزنة أي يمكنك كتابة:

```
EXECUTE getProducts;
```

بالطبع تفتقد الاستعلامات المخزنة في Access إلى الكثير من عناصر الإجرائيات المخزنة في قواعد البيانات الأخرى. ففي الاستعلامات المخزنة لا نستطيع استخدام أكثر من تعبير SQL أي لا نستطيع القيام بأكثر من عملية في استعلام مخزن وحيد.

مثال:

إذا أردنا إنشاء استعلام مخزن بسيط في Access ليقوم بعملية حذف السجلات التي تكون فيها قيمة الحقل Quantity أصغر من قيمة محددة نكتب الصيغة:

```
CREATE PROCEDURE deleteQuantity (@myQuantity INT)
AS
Delete from Products where Quanity <@myQuantity;
```

الآن نستطيع تنفيذ هذا الاستعلام المخزن عن طريق استدعائه بأمر EXECUTE فإذا أردنا حذف جميع السجلات التي قيمة الكميات فيها أقل من 8 نكتب الصيغة:

```
EXECUTE deleteQuantity (8);
```

إنشاء إجرائية مخزنة

إنشاء إجرائية مخزنة في Ms Access:

لا يدعم Access الإجرائيات المخزنة ولكنه يسمح لك بإنشاء استعلامات مخزّنة باستخدام التعبير CREATE PROCEDURE.
كما يُمكنك استعمال اسم الاستعلام المخزن الذي أنشأته (كما هي الحال عند استعمال أسماء المناظير ضمن استعلام).
بالطبع تفتقر الاستعلامات المخزنة في Access إلى الكثير من النقاط الموجودة في الإجرائيات المخزنة في قواعد البيانات الأخرى.
ففي الاستعلامات المخزنة لا نستطيع استخدام أكثر من تعبير SQL أي لا نستطيع القيام بأكثر من عملية في استعلام مخزن وحيد.

حذف إجرائية مخزنة

عملية حذف إجرائية مخزنة:

تستخدم أغلب قواعد البيانات الصيغة المستخدمة لحذف إجرائية مخزنة هي:

```
DROP PROCEDURE procedure_name;
```

عملية تعديل إجرائية مخزنة:

لتعديل إجرائية مخزنة نستخدم التعبير ALTER PROCEDURE وهو مشابه للتعبير CREATE PROCEDURE مع فارق واحد يكمن في أنه يعيد إنشاء الإجرائية المخزنة.

مثال:

إذا أردنا تعديل الإجرائية المخزنة myStoredProcedure التي تقوم بإدراج سجل في جدول myTable لتقوم بحذف سجل من هذا الجدول نكتب الصيغة:

```
ALTER PROCEDURE myStoredProcedure (myRecordID INT)
AS
Delete from myTable where ID = myRecordID;
```

بالطبع يُراعى في الصيغة السابقةالاختلافات المذكورة سابقاً بين قواعد البيانات المختلفة والتي تطابق تلك المستخدمة في CREATE PROCEDURE.

إنشاء واستخدام المتحولات والمعاملات

ذكرنا مسبقاً أن هناك ثلاثة أنواع من المعاملات في الإجرائيات المخزنة هي: معاملات الدخل، معاملات الخرج ومعاملات الدخل/خرج.

تستخدم معاملات الدخل لتمرير البيانات إلى الإجرائية، ومعاملات الخرج لإعادة قيم من الإجرائية، ومعاملات الدخل خرج لتأمين الغرضين معاً.

قبل الدخول في تفاصيل معاملات الدخل والخرج لابد من أن نتكلم عن المتحولات في SQL بصورة عامة.

المتحولات في SQL:

المتحولات في SQL مشابهة للمتحولات في اللغات الأخرى غرضها الأساسي تخزين قيم في الذاكرة يمكن استرجاعها باستخدام اسم المتحول.

للتصريح عن متحول نستخدم الصيغة:

```
DECLARE var_name var_type (length);
```

يمكن استخدام عبارة تصريح واحدة لأكثر من متحول في حال كانت جميع المتحولات المراد التصريح عنها من نفس النوع ونفس الحجم. تأخذ العبارة الصيغة:

```
DECLARE var1_name, var2_name, var3_name var_type (length);
```

مثال:

إذا أردنا التصريح عن متحول صحيح وثلاثة متحولات من نوع varchar نكتب الصيغة:

```
DECLARE var1 INT;
DECLARE var2, var3, var4 varchar(50);
```

في قواعد بيانات DB2 يجب تعريف المتحولات ضمن التركيب BEGIN END.

أما في Oracle فتوضع صيغة التصريح قبل كتلة BEGIN END ويجب أن يتم التصريح عن أي متحول قبل استخدامه في صيغة التصريح تلك.

إنشاء واستخدام المتحولات والمعاملات

ذكرنا مسبقاً أن هناك ثلاثة أنواع من المعاملات في الإجرائيات المخزنة هي: معاملات الدخل ، معاملات الخرج ومعاملات الدخل خرج حيث تستخدم معاملات الدخل لتمرير البيانات إلى الإجرائية ومعاملات الخرج لإعادة قيم من الإجرائية ومعاملات الدخل خرج لتأمين الغرضين معاً.

قبل الدخول في تفاصيل معاملات الدخل والخرج لابد من أن نتكلم عن المتحولات في SQL بصورة عامة.

المتحولات في SQL:

المتحولات في SQL مشابهة للمتحولات في اللغات الأخرى غرضها الأساسي تخزين قيم في الذاكرة يمكن استرجاعها باستخدام اسم المتحول.

يمكن استخدام عبارة تصريح واحدة لأكثر من متحول في حال كانت جميع المتحولات المراد التصريح عنها بعبارة التصريح تلك من نفس النوع ونفس الحجم.

في قواعد بيانات DB2 يجب تعريف المتحولات ضمن التركيب BEGIN END.

أما في Oracle فتوضع صيغة التصريح قبل كتلة BEGIN END ويجب أن يتم التصريح عن أي متحول قبل استخدامه في صيغة التصريح تلك.

إسناد قيمة للمتحول

بعد التصريح عن المتحولات قد تحتاج إلى إسناد قيمة لها أحياناً. تختلف الصيغة الخاصة بإسناد قيمة لمتحول من قاعدة بيانات إلى أخرى.

ففي حالة SQL Server هي من الصيغة:

```
SET @var_name = value;
```

أو

```
SELECT @var_name = value;
```

وفي mySQL وDB2 تستخدم SET أيضاً ولكن بدون وضع إشارة @ قبل أسماء المتحولات وباستبدال المساواة بـ ":=" في حالة قواعد بيانات mySQL وOracle أي تصبح الصيغة:

```
var_name:= value;
```

تستخدم Oracle وDB2 أيضاً طريقة خاصة لإسناد قيمة للمتحولات من استعلام يستخدم تعبير SELECT وذلك بالصيغة:

```
SELECT field_name from tableName Into variable_name
where field_name = 1;
```

ما أن يصبح للمتحول قيمة تستطيع استخدامه بحرية ضمن الاستعلام.

إعطاء قيمة للمتحول

بعد التصريح عن المتحولات قد تحتاج إلى إعطائها قيمة أحياناً. تختلف الصيغة الخاصة بإعطاء قيمة لمتحول من قاعدة بيانات إلى أخرى.

ما أن يصبح للمتحول قيمة تستطيع استخدامه بحرية ضمن الاستعلام.

إسناد قيمة لمتحول

لنعد مجدداً إلى المعاملات الخاصة بالإجرائيات المخزنة.

ماذا لو أردنا أن نعطي قيم تلقائية للمعاملات الخاصة بإجرائية مخزنة؟

لإسناد قيم تلقائية للمعاملات نستخدم إشارة المساواة بعد تعريف المعامل ونضع القيمة التلقائية كما في حالة SQL Server أو باستخدام التعبير Default كما في حالة Oracle.

مثال:

إذا أردنا إنشاء الإجرائية المخزنة التي تقوم بإدراج إسم وعنوان كل شخص في جدول Contacts بحيث يأخذ العنوان القيمة التلقائية 'Unknown'. في حال عدم إعطاء قيمة للمتغير نكتب الصيغة:

```
CREATE PROCEDURE Insert Contacts
(@myName varchar(50), @myAddress varchar(50) = 'Unknown')
AS Insert Into Contacts (contactName, contactAddress)
Values (@myName, @myAddress);
```

أما في Oracle فتكون الصيغة:

```
CREATE OR REPLACE PROCEDURE Insert Contacts
(myName IN varchar(50), myAddress IN varchar(50) DEFAULT 'Unknown')
AS
BEGIN
Insert Into Contacts (contactName, contactAddress)
Values (myName, myAddress);
END;
```

أما قواعد بيانات DB2 فلا تدعم القيمة التلقائية للمعاملات.

إسناد قيمة لمتحول

لنعد مجدداً إلى المعاملات الخاصة بالإجرائيات المخزنة.
ماذا لو أردنا أن نعطي قيم تلقائية للمعاملات الخاصة بإجرائية مخزنة؟
لإعطاء قيم تلقائية للمعاملات نستخدم إشارة المساواة بعد تعريف المعامل ونضع القيمة التلقائية كما في حالة SQL Server أو باستخدام التعبير Default كما في حالة Oracle.

استخدام معاملات الخرج

كما سبق ذكرنا فإن معاملات الخرج مخصصة لإعادة قيم من الإجرائية المخزنة إلى التطبيق الذي نفذ الإجرائية.

استخدام معاملات الخرج في SQL Server:

لتحديد كون أحد المعاملات في إجرائية مخزنة في SQL Server كمعامل خرج، نضيف التعبير OUTPUT وتكون الصيغة من الشكل:

```
CREATE PROCEDURE procedure_name
(@output_parameter_name INT OUTPUT)
…;
```

مثال:

إذا أردنا إنشاء إجرائية مخزنة تقوم بإعادة اسم شخص صاحب رقم هاتف معين ووضع القيمة المعادة في متحول الخرج theName نكتب الصيغة:

```
CREATE PROCEDURE getName
(@theNumber varchar(15) , @theName varchar(50) OUTPUT)
AS
BEGIN
SET @theName = (SELECT Name from Phonebook where Number = @theNumber)
END;
```

نلاحظ أننا استخدمنا التعبير SET لنعطي المتحول theName القيمة التي أعادها الاستعلام.

ويمكننا استخدام هذه الإجرائية المخزنة مباشرةً بالشكل:

```
DECLARE @theName varchar(50);
EXECUTE getName '4445467' , @theName OUTPUT;
PRINT @theName;
```

قمنا هنا:

- بالتصريح عن المتحول theName،
- ثم بتنفيذ الإجرائية المخزنة مستخدمين theName كمتحول خرج لاستلام القيمة التي تعيدها الإجرائية،
- ثم بإخراج قيمة هذا المتحول.

استخدام معاملات الخرج

كما ذكرنا فإن معاملات الخرج مخصصة لإعادة قيم من الإجرائية المخزنة إلى التطبيق الذي نفذ الإجرائية.

استخدام معاملات الخرج في SQL Server:

لتحديد كون أحد المعاملات في إجرائية مخزنة في SQL Server كمعامل خرج نضيف التعبير OUTPUT.

في مثالنا هنا، قمنا:

- بالتصريح عن المتحول theName،
- ثم بتنفيذ الإجرائية المخزنة مستخدمين theName كمتحول خرج لاستلام القيمة التي تعيدها الإجرائية،
- ثم بإخراج قيمة هذا المتحول.

استخدام معاملات الخرج

استخدام معاملات الخرج في Oracle:

إذا أردنا كتابة نفس المثال السابق ولكن باستخدام قواعد بيانات Oracle ستكون الصيغة كالتالي:

```
CREATE OR REPLACE PROCEDURE getName
(theNumber IN varchar(15) , theName OUT varchar(50))
AS
BEGIN
SELECT Name INTO theName from Phonebook
where Number = theNumber;
END;
```

نلاحظ هنا استخدام التعبير SELECT INTO لتخزين قيمة Name في الحقل theName.

إذا أردنا تنفيذ هذه الإجرائية المخزنة نكتب الصيغة:

```
SET SERVER ON
DECLARE theName varchar(50);
BEGIN
getName ('4465873' , theName);
dbms_output.put_line(theName);
END;
```

في هذه الحالة:

- استخدمنا التعبير SET SERVER ON وهو أمر SQL* plus يسمح لنا بتمرير الخرج إلى SQL* plus
- ثم صرحنا عن المتحول theName
- ثم استدعينا الإجرائية التي أعادت بدورها القيمة المطلوبة إلى المتحول theName
- ثم طبعنا المتحول.

استخدام معاملات الخرج

استخدام معاملات الخرج في Oracle

في هذه الحالة:

- استخدمنا التعبير SET SERVER ON وهو أمر SQL* plus يسمح لنا بتمرير الخرج إلى SQL* plus

- ثم صرحنا عن المتحول theName
- ثم استدعينا الإجرائية التي أعادت بدورها القيمة المطلوبة إلى المتحول theName
- ثم طبعنا المتحول.

استخدام معاملات الخرج

استخدام معاملات الخرج في DB2:

لكتابة نفس المثال السابق باستخدام قواعد بيانات DB2 نستخدم الصيغة:

```
CREATE PROCEDURE getName
(theNumber INT , OUT theName varchar(50))
P1:BEGIN
SET theName = (SELECT Name from Phonebook where Number = theNumber);
END P1;
```

كما نلاحظ قمنا بإعطاء المتغير theName قيمة معادة من الاستعلام باستخدام التعبير SET فإذا أردنا الآن استدعاء الإجرائية المخزنة نستخدم الصيغة:

```
CALL (getName '5738894' , ?);
```

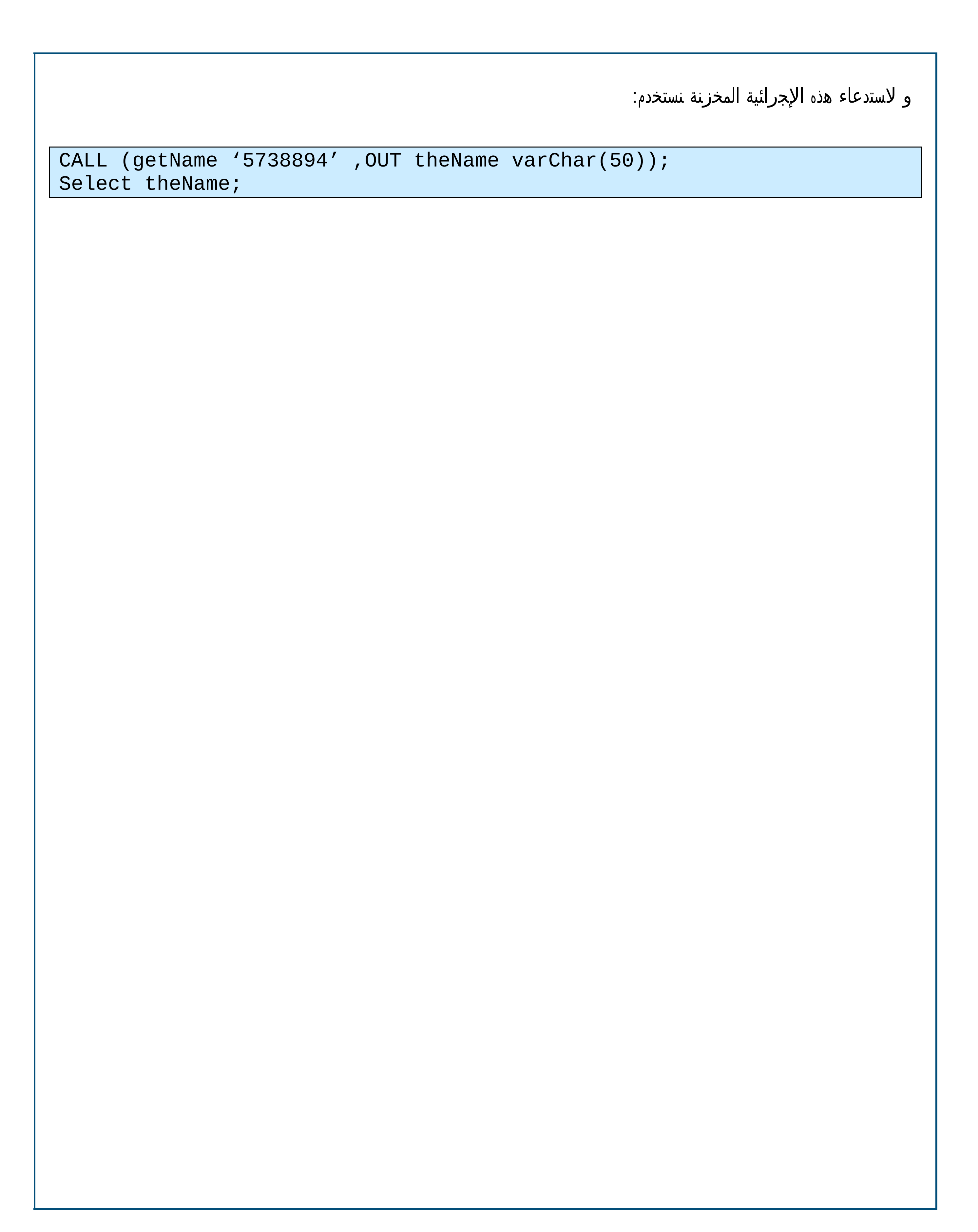
استخدمت هنا إشارة الاستفهام كبديل عن معامل الخرج مما يمكن DB2 من إظهار القيمة التي يعيدها هذا المعامل.

استخدام معاملات الخرج في MySQL:

لكتابة نفس المثال في Mysql نستخدم الصيغة:

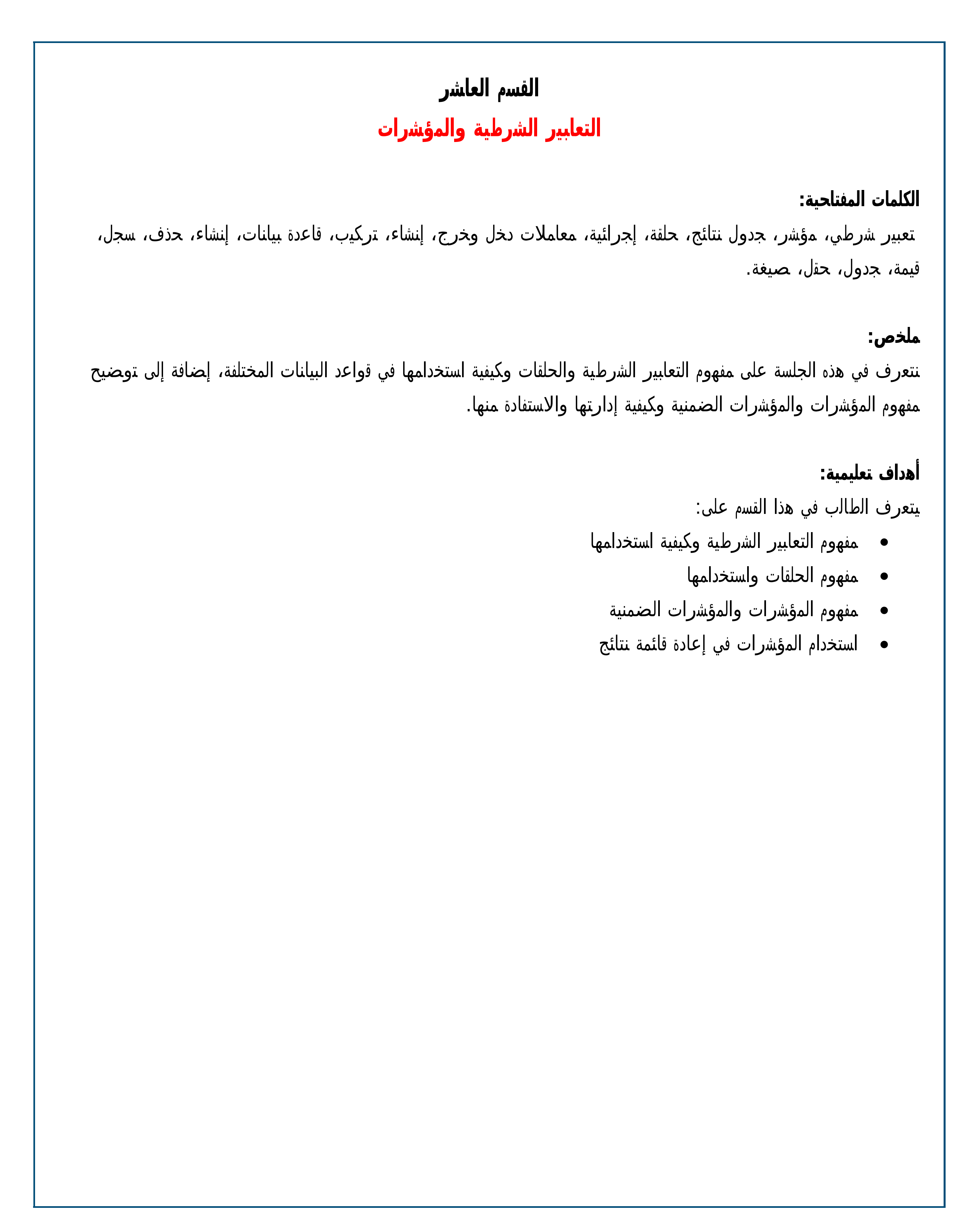
```
CREATE PROCEDURE getName
(IN theNumber varchar(15) , OUT theName varchar(50))

BEGIN
SELECT Name INTO theName from Phonebook
where Number = theNumber;
END;
```



و لاستدعاء هذه الإجرائية المخزنة نستخدم:

```
CALL (getName '5738894' ,OUT theName varChar(50));
Select theName;
```



القسم العاشر

التعابير الشرطية والمؤشرات

الكلمات المفتاحية:

تعبير شرطي، مؤشر، جدول نتائج، حلقة، إجرائية، معاملات دخل وخرج، إنشاء، تركيب، قاعدة بيانات، إنشاء، حذف، سجل، قيمة، جدول، حقل، صيغة.

ملخص:

نتعرف في هذه الجلسة على مفهوم التعابير الشرطية والحلقات وكيفية استخدامها في قواعد البيانات المختلفة، إضافة إلى توضيح مفهوم المؤشرات والمؤشرات الضمنية وكيفية إدارتها والاستفادة منها.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- مفهوم التعابير الشرطية وكيفية استخدامها
- مفهوم الحلقات واستخدامها
- مفهوم المؤشرات والمؤشرات الضمنية
- استخدام المؤشرات في إعادة قائمة نتائج

الإجرائيات المخزنة

كتابة النصوص البرمجية في الإجرائيات المخزنة:

ذكرنا مسبقاً أنّ الإجرائيات المخزنة تتألف بصورة أساسية من تعابير SQL بالإضافة إلى لغة برمجة بنيوية تتضمن تعابير خاصة للتنفيذ الشرطي والحلقات وإمكانيات أخرى.

تختلف التعابير البرمجية بين أنظمة قواعد البيانات ولكنها تتشابه في التعليمات الأساسية.

التنفيذ الشرطي:

في الكثير من الأحيان نجد أنه من الضروري ألا يتم تنفيذ استعلام ما ضمن إجرائية مخزنة إلا عند تحقق شرط معين كإدخال معامل بقيمة صحيحة مثلاً.

قد يكون أحد السيناريوهات المحتملة أيضاً الرغبة في عدم تنفيذ استعلام ما لم يعيد استعلام سابق نتيجة متوقعة.

أو أن نحتاج إلى تنفيذ استعلامات مختلفة بحسب قيمة معامل محدد مثلاً.

تحتوي أغلب أنظمة قواعد البيانات العلائقية على نوعين أساسين من التعابير الشرطية:

- التعبير IF..........ELSE وهو بالصيغة:

```
IF condition
conditionTrueBody;
ELSE
conditionFalseBody;
```

حيث تمثل condition الشرط الواجب تحققه لتنفيذ العبارات التي تمثلها conditionTrueBody وإلا سيتم تنفيذ العبارات التي تمثلها conditionFalseBody.

- التعبير WHEN............CASE ويكتب بالصيغة:

```
CASE expression
WHEN value1 THEN result1
WHEN value2 THEN result2
…..
WHEN valueN THEN resultN
ELSE resultElse
END;
```

في هذه الصيغة سيتم تقييم التعبير expression ومقارنته مع القيم Value1…..ValueN وستكون نتيجة هذه الصيغة هي النتيجة

result المقابلة لأول عملية مطابقة، وإلا سيتم تنفيذ النتيجة resultElse.

سنقوم فيمايلي باستعراض عمل هذه البنى واختلافات تطبيقها بين قواعد البيانات المختلفة.

الإجرائيات المخزنة 1

كتابة النصوص البرمجية في الإجرائيات المخزنة:

ذكرنا مسبقاً أنّ الإجرائيات المخزنة تتألف بصورة أساسية من تعبيرات SQL بالإضافة إلى لغة برمجة بنيوية تتضمن تعبيرات خاصة للتنفيذ الشرطي والحلقات وإمكانيات أخرى.

تختلف التعبيرات البرمجية بين أنظمة قواعد البيانات ولكنها وتتشابه في التعليمات الأساسية.

التنفيذ الشرطي:

في الكثير من الأحيان نجد أنه من الضروري ألا يتم تنفيذ استعلام ما ضمن إجرائية مخزنة إلا عند تحقق شرط معين كإدخال معامل بقيمة صحيحة مثلاً.

قد يكون أحد السيناريوهات المحتملة أيضاً الرغبة في عدم تنفيذ استعلام ما لم يعيد استعلام سابق نتيجة متوقعة.

أو أن نحتاج إلى تنفيذ استعلامات مختلفة بحسب قيمة معامل محدد مثلاً.

تحتوي أغلب أنظمة قواعد البيانات العلائقية على نوعين أساسين من التعبيرات الشرطية:

ا لتعبير IF..........ELSE

التعبير CASE............WHEN

سنقوم فيمايلي باستعراض عمل هذه البنى واختلافات تطبيقها بين قواعد البيانات المختلفة

التعابير الشرطية

سنرى كيفية استثمار التعابير الشرطية IF...ELSE وCASE...WHEN في قواعد بيانات SQL Server ونحدد بعد ذلك الاختلافات في حال استخدام أنواع قواعد البيانات الأخرى.

التركيب IF.....ELSE:

تأخذ بنية IF….ELSE الشرطية الصيغة التالية في SQL Server:

```
IF condition
BEGIN
trueStatments
END
ELSE
BEGIN
falseStatments
END
```

يمكن حذف BEGIN وEND من الصيغة السابقة في حال كان المطلوب تنفيذ تعبير واحد في كتلة BEGIN END.

كما يمكن كذلك الاستغناء عن ELSE مع الكتلة الخاصة بها في حال عدم الرغبة بتنفيذ أي تعبير عند عدم تحقق الشرط.

مثال:

إذا أردنا أن نكتب الإجرائية المخزنة التي تقوم بحساب كمية المبيعات من مادة ما، وكتابة تعليق في حقل خاص في جدول المواد عن ضرورة إيقاف هذه السلعة في حال عدم بيع أي كمية منها، نكتب الصيغة:

```
CREATE PROCEDURE getSalesAndComment
(@myProductID INT, @mySales INT OUTPUT,@myComment VARCHAR(40) OUTPUT)
AS
BEGIN

SET @mySales= (Select sum(Quantity) from sales
 where productID=@myProductID);
IF @mySlales=0
SET @myComment='STOP THIS PRODUCT';
ELSE
SET @myComment='KEEP THIS PRODUCT';
Update products set ProductComment=@myComment
Where productID=@myProductID ;
END
```

نلاحظ أننا لم نستخدم كتلة BEGIN END لأننا لم نطلب تنفيذ سوى أمر واحد ضمن كتلة IF وELSE، لاستخدام هذه الإجرائية يمكننا كتابة الصيغة:

```
DECLARE @theComment VARCHAR(40);
DECLARE @theSales INT
EXECUTE getSalesAndComment(3, @theSales OUTPUT, @theComment OUTPUT);
PRINT @theSales;
PRINT @theComment;
```

نلاحظ أننا صرحنا عن المتغيرات التي سيتم إعادتها من الإجرائية وقمنا بطباعتها.

التعبيرات الشرطية

سنرى كيفية استثمار التعبيرات الشرطية IF…ELSE وCASE…WHEN في قواعد بيانات SQL Server ونحدد بعد ذلك الاختلافات في حال استخدام أنواع قواعد البيانات الأخرى.

التعبيرات الشرطية

التركيب CASE….WHEN:

تأخذ بنية CASE…WHEN الشرطية الصيغة التالية في SQL Server:

```
SET @aVariable= CASE expression
WHEN value1 THEN result1
WHEN value2 THEN result2
…..
WHEN valueN THEN resultN
ELSE resultElse
END;
```

لا نستطيع استخدام التركيب CASE…WHEN في SQL Server إلا ضمن عبارة SELECT أو عبارة [Yskh قيمة لمتغير.
يمكن استخدام التعبير CASE دون وضع أي تعبير بعدها حيث يمكن هنا استخدام value1…valueN لوضع شروط مختلفة.

مثال:

إذا أردنا أن نكتب الإجرائية المخزنة التي تقوم بعرض عبارة تعتمد على علامة طالب وتعبر عن مستواه بعد أن تأخذ على الدرجة التي حصل عليها:

```
CREATE PROCEDURE getStudentLevel
(@myStudentName VARCHAR(50), @myStudentLevel VARCHAR(40) OUTPUT)
AS
BEGIN
DECLARE studentGrade INT;
SET @studentGrade= (Select studentGrade from students
 where studentName=@myStudentName);
SET @myStudentLevel= CASE
WHEN @studentGrade>80 THEN 'VERY GOOD'
WHEN @studentGrade>70 THEN 'GOOD'
WHEN @studentGrade>60 THEN 'NOT BAD'
WHEN @studentGrade>50 THEN 'PASS'
WHEN @studentGrade<50 THEN 'FAIL'
END;
END;
```

و لاستخدام هذه الإجرائية يمكننا كتابة الصيغة:

```
DECLARE @theStudentLevel VARCHAR(40);
EXECUTE getSalesAndComment('sami', @ theStudentLevel OUTPUT);
PRINT @ theStudentLevel;
```

هنا كما نلاحظ صرحنا عن المتغيرات التي سيتم إعادتها من الإجرائية وقمنا بطباعتها.

التعبيرات الشرطية

يتم تطبيق التراكيب IF…ELSE وCASE…WHEN في Oracle وDB2 وmySQL بصورة مشابهة لتلك المستخدمة في SQL Server مع الانتباه إلى الاختلافات التالية:

- في Oracle وDB2 وmySQL يمكنك استخدام IF…ELSE في التركيب IF…ELSE لتقييم شروط إضافية.
- في Oracle وDB2 وmySQL يمكن لـ CASE…WHEN أن تُستخدم بصورة منفصلة وليست بحاجة لأن تكون جزء من عبارة إسناد قيمة لمتغير، أو جزء من عبارة SELECT كما هي الحال في SQL Server.
- يتم إغلاق التركيب CASE…WHEN بالتعبير END CASE وليس END كما هي الحال في SQL Server.

- بالطبع لن نغفل الاختلافات المذكورة مسبقاً كضرورة تحويل إشارات المساواة إلى إشارات:= في Oracle عوضاً عن استخدام SET.

مثال:

لنكتب الإجرائية المخزنة التي تقوم بإدراج حجز جديد ضمن جدول الحجوزات Reservations وحساب كلفة الحجز بناء على عدد الأيام ونوع الغرفة، بصيغة قواعد بيانات Oracle:

```
CREATE OR REPLACE PROCEDURE Reservations
(myRoomType IN varchar(10), myDays IN INT)
AS
BEGIN
IF myRoomType= 'Single' THEN
Insert Into Reservations (roomType, Fee)
Values(myRoomType,20*myDays);
ELSEIF myRoomType='Double' THEN
Insert Into Reservations (roomType, Fee)
Values(myRoomType,35*myDays);
ELSEIF myRoomType='Sweet' THEN
Insert Into Reservations (roomType, Fee)
Values(myRoomType,45*myDays);
END IF;
END;
```

و لاستخدام هذه الإجرائية يمكننا كتابة الصيغة:

```
CALL Reservations ('Single',5);
```

الحلقات والتكرار

تدعم قواعد البيانات تعابير مختلفة لتأمين تنفيذ كتلة من التعليمات لأكثر من مرة اعتماداً على استمرار تحقق شرط ما.

ففي SQL Server يستخدم التعبير WHILE مع BEGIN وEND بالصيغة:

```
WHILE Condition
BEGIN
loopBody
END;
```

أما في Oracle فيُستخدم لتحديد بداية جسم الحلقة التعبير LOOP ولنهايتها END LOOP وذلك كما يظهر في الصيغة:

```
WHILE Condition
LOOP
loopBody
END LOOP;
```

وفي DB2 نستخدم التعبير DO لتحديد جسم الحلقة والتعبير END WHILE لتحديد نهايتها وذلك حسب الصيغة:

```
WHILE Condition
DO
loopBody
END WHILE;
```

وفي mySQL يُحدد جسم الحلقة بالتعبير WHILE ونهايتها بالتعبير END WHILE حسب الصيغة التالية:

```
WHILE Condition
loopBody
END WHILE;
```

مثال:

إذا أردنا كتابة الإجرائية المخزنة التي تقوم بتوليد عشر أرقام عشوائية بين 1 و100 ووضعها في الجدول Numbers وذلك في قواعد بيانات SQL Server نستخدم الصيغة:

```
CREATE PROCEDURE tenRandoms()
AS
BEGIN
DECLARE @myNumber INT;
WHILE @myNumber<10
BEGIN
Insert Into Numbers(number) Values(Round(rand)*100);
SET @myNumber=@myNumber+1;
END;
END;
```

الحلقات والتكرار

تدعم قواعد البيانات تعبيرات مختلفة لتأمين تنفيذ كتلة من التعليمات لأكثر من مرة اعتماداً على استمرار تحقق شرط ما.
ففي SQL Server يستخدم التعبير WHILE مع BEGIN وEND.

أما في Oracle فيُستخدم لتحديد بداية جسم الحلقة التعبير LOOP ولنهايتها END LOOP.

وفي mySQL يُحدد جسم الحلقة بالتعبير WHILE ونهايتها بالتعبير END WHILE

الحلقات والتكرار

إذا أردنا تطبيق نفس المثال السابق على قواعد بيانات Oracle ستكون الصيغة:

```
CREATE OR REPLACE PROCEDURE tenRandoms()
AS
BEGIN
DECLARE myNumber INT;
myNumber:=1;
WHILE myNumber<10
LOOP
Insert Into Numbers(number) Values(Round(rand)*100);
MyNumber:=myNumber+1;
END LOOP;
END;
```

وفي DB2 تصبح الصيغة:

```
CREATE PROCEDURE tenRandoms()
AS
P1:BEGIN
DECLARE myNumber INT;
myNumber=1;
WHILE myNumber<10
DO
Insert Into Numbers(number) Values(Round(rand)*100);
myNumber=myNumber+1;
END WHILE;
END P1;
```

وفي mySQL تصبح الصيغة:

```
CREATE PROCEDURE tenRandoms()
BEGIN
DECLARE myNumber INT;
SET myNumber=1;
WHILE myNumber<10
Insert Into Numbers(number) Values(Round(rand)*100);
SET myNumber=myNumber+1;
END WHILE;
END;
```

استخدام المؤشرات

ما هي المؤشرات؟

المؤشرات هي عبارة عن طريقة لتمثيل مجموعة سجلات ضمن نص SQL بحيث تسمح بالدوران على هذه السجلات.

بصورة عامة، تؤثر المؤشرات على أداء قاعدة البيانات وينصح عادة بعدم استخدامها إلا في حال الضرورة، وبالأخص عند الحاجة إلى القدرة على التحرك بين كل سجل من مجموعة من السجلات بصورة منفصلة.

التصريح عن مؤشر:

للتصريح عن مؤشر نستخدم الصيغة:

```
DECLARE cursorName CURSOR FOR cursorSpecification;
```

حيث تعبر cursorSpecification عن الاستعلام الذي سوف يستخدَم معه المؤشر.

في Oracle نستخدم IS عوضاً عن FOR في صيغة التصريح عن مؤشر.

في حال أردنا تعديل البيانات عن طريق المؤشر توجب علينا إضافة FOR UPDATE إلى نهاية الصيغة حيث تصبح الصيغة:

```
DECLARE cursorName CURSOR IS cursorSpecification FOR UPDATE;
```

تقوم هذه العملية بإقفال الجداول التي يستخدمها المؤشر بحيث لا يتمكن مستخدم آخر من تعديلها.

يمكننا تحديد الجداول التي نود إقفالها بإضافة OF columnList حيث columnList هي قائمة بأسماء الحقول وفي هذه الحالة سيتم فقط إقفال الجداول التي تنتمي إليها تلك الحقول.

يمكننا أيضاً تحديد الخيار FOR READ ONLY عوضاً عن FOR UPDATE إذا كنا لا نريد للمؤشر أن يكون قابلاً للتعديل.

وفي قواعد بيانات DB2 هناك خياران إضافيان هما WITH HOLD وWITH RETURN.
يمكّن الخيار WITH HOLD من تحديد كون بقاء المؤشر مفتوحاً لأكثر من مناقلة.
أما الخيار WITH RETURN فيمكن أن يكون من الشكل TO CALLER وفي هذه الحالة يمكن إعادة المؤشر إلى أي تطبيق أو أي إجرائية مخزنة أخرى أو الشكل TO CLIENT حيث يسمح بإعادة المؤشر مباشرة إلى التطبيق الزبون ولن يكون مرئياً لأي من الإجرائيات المخزنةالوسيطة.

استخدام المؤشرات

ما هي المؤشرات؟
المؤشرات هي عبارة عن طريقة لتمثيل مجموعة سجلات ضمن نص SQL بحيث تسمح بالدوران على هذه السجلات.
بصورة عامة، تؤثر المؤشرات على أداء قاعدة البيانات وينصح عادة بعدم استخدامها إلا في حال الضرورة، وبالأخص عند الحاجة إلى القدرة على التحرك بين كل سجل من مجموعة من السجلات بصورة منفصلة.

استخدام المؤشرات

قبل استخدام مؤشر علينا أولاً أن نقوم بفتح هذا المؤشر وذلك بالصيغة:

```
OPEN cursorName;
```

تستطيع الآن بعد فتح المؤشر استعادة سجلات منه وتخزينها في متحولات محلية.
للقيام بهذا العمل سنستخدم التعبير FETCH وصيغته من الشكل:

```
FETCH cursorName INTO var1,var2,var3,…,varN;
```

تقوم هذه العملية بإعطاء المتحولات var1,var2,var3,…,varN قيم حقول السجل الحالي للمؤشر.
يجب أن تتطابق أنواع هذه المتحولات مع أنواع الحقول الموافقة لها.

الصيغة السابقة هي صيغة DB2. تختلف الأمور قليلاً بالنسبة لـ SQL Server حيث تصبح من الشكل:

```
FETCH NEXT from cursorName INTO @var1,@var2,@var3,…,@varN;
```

يمكننا التنقل للحصول على قيم الحقول للسجل الأول باستخدام FETCH FIRST، وللسجل السابق باستخدام FETCH PRIOR، وللسجل الأخير باستخدام FETCH LAST عوضاً عن FETCH NEXT.

كما يمكننا تحديد رقم السجل الذي نود استعادة معلوماته وذلك باستخدام FETCH ABSOLUTE N أو FETCH RELATIVE N.

تصبح المرحلة الأخيرة بعد فتح المؤشر ونقل معلومات السجل إلى المتحولات الخاصة به، هي إغلاق المؤشر وذلك باستخدام الصيغة:

```
CLOSE cursorName;
```

انتبه:
قد تحتاج في SQL Server إلى تحرير الموارد التي استخدمها مؤشرك.

المؤشرات الضمنية

هناك طريقة في قواعد بيانات Oracle وDB2 لإنشاء المؤشر ولفتحه، وللدوران باستخدامه ومن ثم إغلاقه.
تستخدم هذه الطريقة التعبير FOR للتحرك ضمن سجلات المؤشر.
الصيغة في Oracle هي التالية:

```
FOR cursorName IN (cursorSpecification)
LOOP
loopBody
END LOOP;
```

أما في DB2 فتكون الصيغة:

```
FOR cursorName AS (cursorSpecification)
DO
loopBody
END FOR;
```

سوف تقوم هذه الصيغ بفتح المؤشر، وبالدوران في السجلات التي يحويها المؤشر كل بدوره ثم تغلق المؤشر.

مثال:

إذا أردنا كتابة الصيغة التي تقوم بطباعة جميع قيم الحقل myColumn من جدول myTable عن طريق استخدام المؤشرات نستخدم الصيغة:

```
SET SERVEROUT ON
FOR myCursor IN (select myColumn from myTable)
LOOP
Dbms_output.put_line(myCursor.myColumn)
END LOOP;
```

قمنا هنا باستخدام المؤشر الضمني myCursor للدوران ضمن مجموعة السجلات المحدد بالاستعلام، وطباعة قيمة الحقل للعمود myColumn. نلاحظ بأننا تمكنا من الوصول إلى القيم باستخدام اسم المؤشر ثم اسم الحقل مفصولاً بنقطة.

استخدام المؤشرات

مثال على استخدام المؤشرات في SQL Server:

نريد كتابة الصيغة التي تقوم بالدوران على مجموعة قيم لأسماء وأرقام هواتف أشخاص وطباعة هذه المعلومات عن طريق استخدام المؤشرات.

```
DECLARE @myNumber varchar(15);
DECLARE @myName varchar(50);
DECLARE myCursor CURSOR FOR
Select contactNumber, contactName from Contacts;
OPEN myCursor;
WHILE @@FETCH_STATUS = 0
FETCH NEXT from myCursor INTO @myNumber, @myName;
PRINT @myName, @myNumber;
END;
```

هنا قمنا بالتصريح عن المؤشر myCursor وتحديد الاستعلام الذي يمثل مجموعة السجلات التي يعبر عنها المؤشر، ثم قمنا بفتح المؤشر والمرور على السجلات واحداً تلو الآخر باستخدام التعبير FETCH NEXT.
استخدمنا هنا @@FETCH_STATUS لتحديد فيما إذا تم الوصول إلى نهاية المؤشر حيث ستكون قيمتها 1- مقابل 0 في الحالة العادية.

مثال على استخدام المؤشرات في Oracle:

إذا أردنا إعادة نفس هذا المثال في قواعد بيانات Oracle تصبح الصيغة:

```
SET SERVEROUT ON
DECLARE myNumber varchar(15);
DECLARE myName varchar(50);
DECLARE myCursor CURSOR IS
Select contactNumber, contactName from Contacts;
OPEN myCursor;
WHILE myCursor%FOUND
LOOP
FETCH NEXT from myCursor INTO myNumber, myName;
dbms_output.put_line(myName, myNumber);
END LOOP;
```

استخدمنا SET SERVEROUT لتمكين الطباعة إلى الخرج.

نلاحظ أننا استخدمنا التعبير myCursor%FOUND. ندعو %FOUND صفة المؤشر حيث يعيد هذا التركيب القيمة TRUE في حال إيجاد السجل الحالي للمؤشر.

توجد صفات أخرى يمكن استخدامها مثل %ISOPEN التي تحدد كون المؤشر مفتوحاً أو مغلقاً، و %ROWCOUNT التي تحدد عدد السجلات التي تم المرور عليها من المؤشر حتى هذه اللحظة.

إعادة جدول نتائج من إجرائية

حتى الآن استعرضنا كيفية استعادة قيم إفرادية من إجرائيات مخزنة باستخدام معاملات الخرج، ولكن يمكن لنا إعادة كامل جدول القيم من إجرائية. تعتبر هذه العملية بسيطة جداً في SQL Server وAccess فهي تأخذ الصيغة:

```
CREATE PROCEDURE procedureName AS query;
```

ويمكننا تنفيذها باستخدام EXEC أو EXECUTE بالصيغة:

```
EXEC procedureName;
```

مثال:

لإنشاء إجرائية باسم myProcedure تقوم بإعادة أسماء المنتجات من جدول المنتجات نكتب الصيغة:

```
CREATE PROCEDURE myProcedure
AS select productName from Products;
```

أما في DB2 فكما نوهنا سابقاً يمكننا إعادة أكثر من جدول واحد اعتماداً على الصيغة:

```
CREATE PROCEDURE procedureName()
RESULT SETS 1
LANGUAGE SQL
P1:BEGIN
DECLARE cursorName CURSOR WITH RETURN FOR
query
open cursorName
END P1
```

نلاحظ أننا أضفنا التعبير RESULT SETS 1 لإعادة جدول نتائج وحيد بحيث يكون مصدر هذا الجدول الاستعلام المعرف ضمن المؤشر cursorName.

يمكن لـ DB2 إعادة أكثر من قائمة نتائج وتعاد بحسب ترتيب فتح المؤشر الخاص بها ولكننا بالطبع نحتاج إلى تعديل قيمة العدد الصحيح الذي يعبر عن عدد جداول النتائج المعادة المستخدم بعد التعبير RESULT SETS.

نلاحظ أيضاً استخدامنا لـ WITH RETURN لأننا نريد للمؤشر أن يعيد جدول القيم من الإجرائية.

لاستدعاء هذه الإجرائية المخزنة في قواعد بيانات DB2 نستخدم الصيغة:

```
CALL procedureName;
```

إعادة جدول نتائج من إجرائية

تكلمنا عن كيفية إعادة جدول قيم من إجرائية مخزنة في قواعد بيانات SQL Server وDB2 أما في Oracle فيتطلب هذا العمل خطوات إضافية. إذ لابد لنا أولاً من استخدام ما يسمى طرد وذلك باستخدام التعبير CREATE PACKAGE حيث يشبه الأمر تعريف نوع جديد من أنماط البيانات وهو نوع يعيد قائمة نتائج.

لإنشاء طرد نكتب الصيغة:

```
CREATE [OR REPLACE] PACKAGE packageName
AS types, procedure, ect…
END packageName;
```

بعد هذا يجب علينا تعريف جسم الطرد وذلك بالتعبير CREATE PACKAGE BODY وفي هذه المرحلة نقوم بتعريف مهمة الطرد:

```
CREATE [OR REPLACE] PACKAGE BODY packageName
AS
definition of procedure
END packageName;
```

يجب أن يكون اسم الطرد واسم جسم الطرد متطابقين.

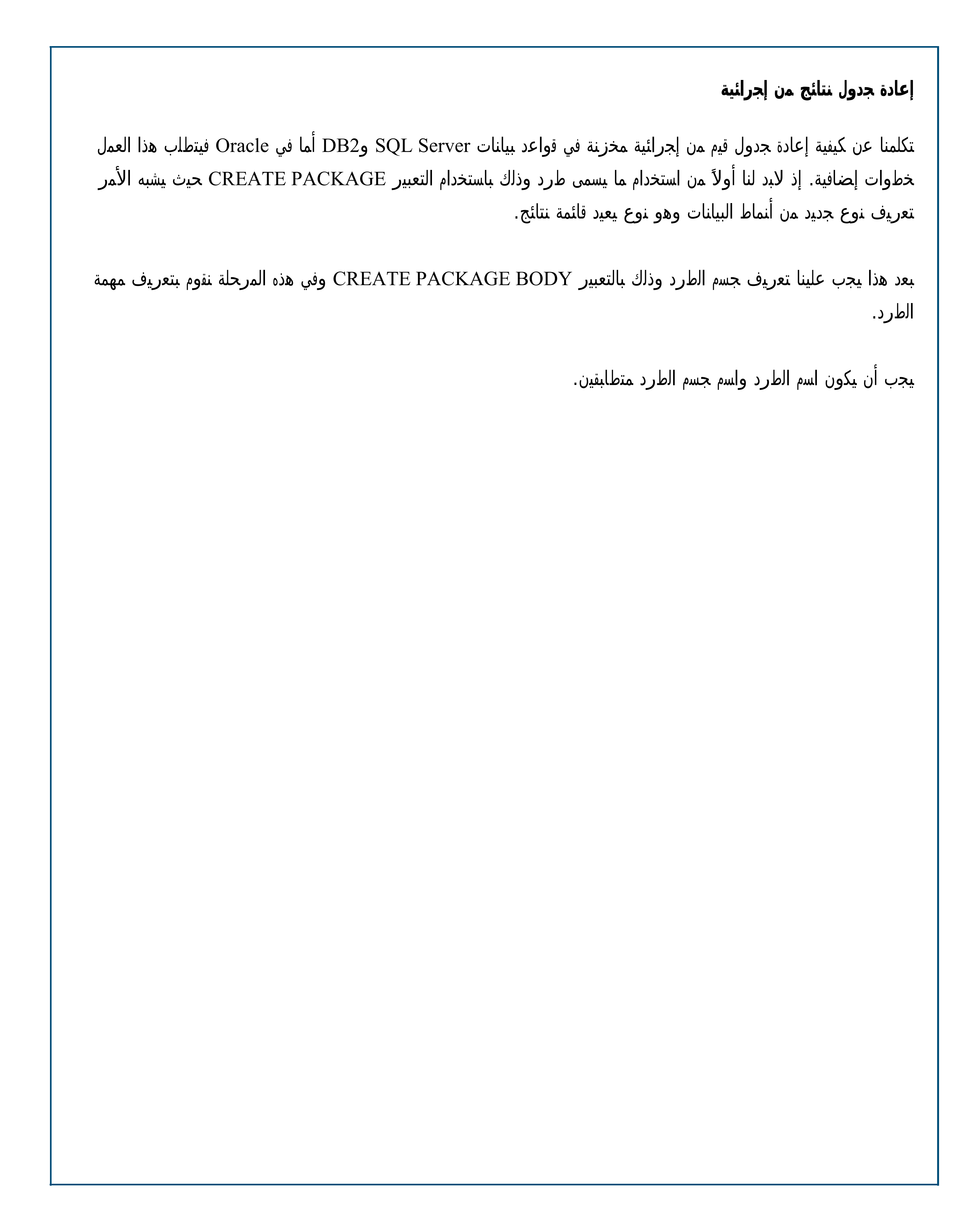
مثال:

إذا أردنا إنشاء طرد باسم myPackage يعرف نمط myCursor وإجرائية myProcedure أحد معاملات الخرج فيها هو theCursor وهو من النمط CURSOR.

```
CREATE OR REPLACE PACKAGE myPackage
AS
Type myCursor IS REF CURSOR;
PROCEDURE myProcedure (theCursor OUT myCursor);
END myPackage;
```

بعدها نقوم بتعريف جسم الطرد بحيث يحمل نفس الاسم ويوضح ماذا يجب أن تفعل الإجرائية وذلك بالصيغة:

```
CREATE OR REPLACE PACKAGE BODY myPackage
AS
PROCEDURE myProcedure (theCursor OUT myCursor);
IS
BEGIN
OPEN theCursor FOR
Select myColumn from myTable;
END myProcedure;
END myPackage;
```

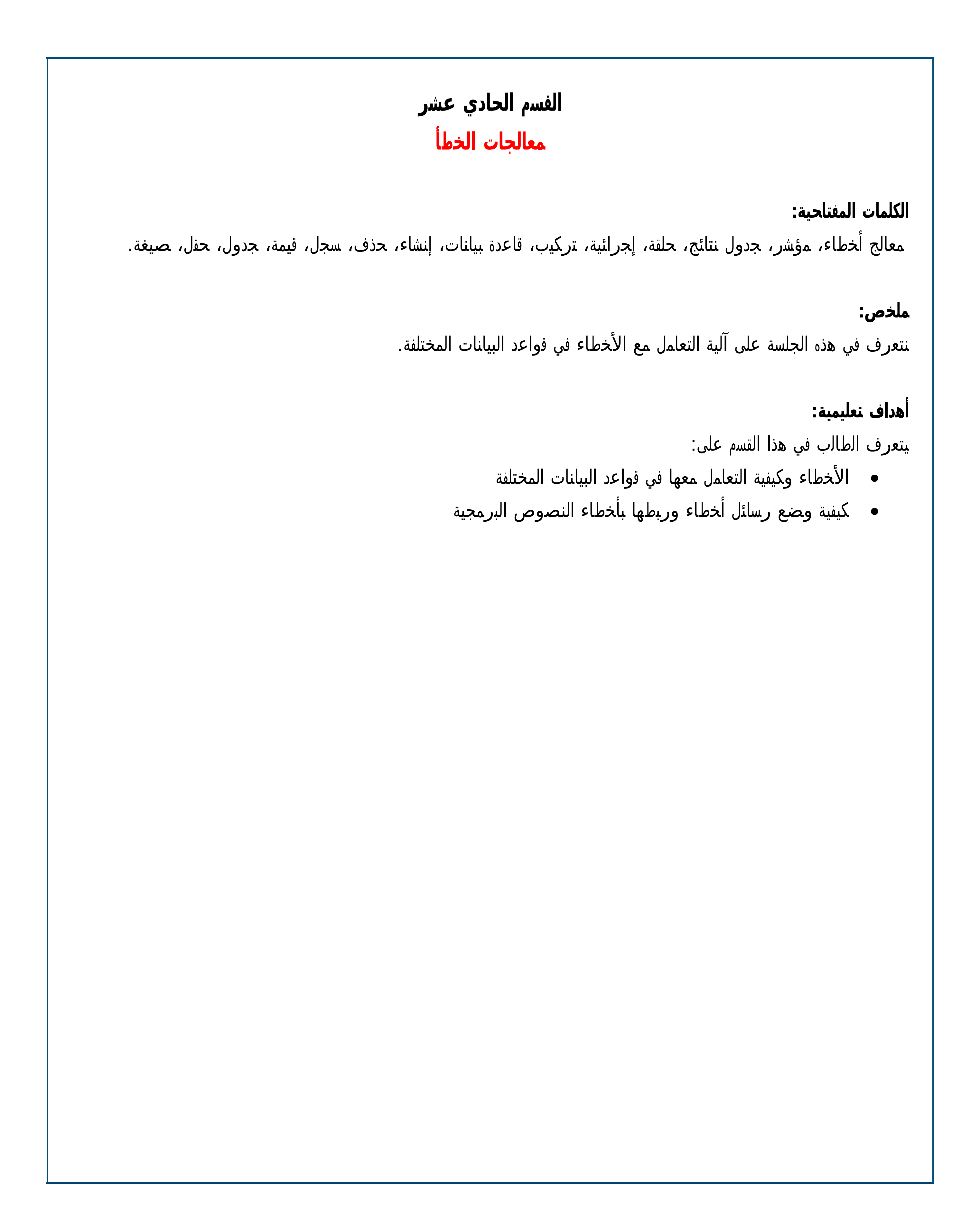


إعادة جدول نتائج من إجرائية

تكلمنا عن كيفية إعادة جدول قيم من إجرائية مخزنة في قواعد بيانات SQL Server وDB2 أما في Oracle فيتطلب هذا العمل خطوات إضافية. إذ لابد لنا أولاً من استخدام ما يسمى طرد وذلك باستخدام التعبير CREATE PACKAGE حيث يشبه الأمر تعريف نوع جديد من أنماط البيانات وهو نوع يعيد قائمة نتائج.

بعد هذا يجب علينا تعريف جسم الطرد وذلك بالتعبير CREATE PACKAGE BODY وفي هذه المرحلة نقوم بتعريف مهمة الطرد.

يجب أن يكون اسم الطرد واسم جسم الطرد متطابقين.



القسم الحادي عشر

معالجات الخطأ

الكلمات المفتاحية:

معالج أخطاء، مؤشر، جدول نتائج، حلقة، إجرائية، تركيب، قاعدة بيانات، إنشاء، حذف، سجل، قيمة، جدول، حقل، صيغة.

ملخص:

نتعرف في هذه الجلسة على آلية التعامل مع الأخطاء في قواعد البيانات المختلفة.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- الأخطاء وكيفية التعامل معها في قواعد البيانات المختلفة
- كيفية وضع رسائل أخطاء وربطها بأخطاء النصوص البرمجية

معالجة الأخطاء

كما ذكرنا تتجاوز الإجرائيات المخزنة حدود الاستعلامات البسيطة وتدخل في إطار لغة برمجة إجرائية.
في مثل هذه البيئة، يعتبر ظهور الأخطاء أحد الاحتمالات الواردة.

قد يتسبب بالخطأ استعلام خاطئ، أو أي عبارة SQL خاطئة، أو قد يكون نتيجة تمرير بيانات غير مناسبة إلى إجرائية مخزنة أو قد يكون أحياناً خطأ مفتعل قمنا بتوليده لسبب ما.

بعكس الأخطاء في بعض لغات البرمجة الأخرى لا تسبب أخطاء SQL بالضرورة بإيقاف تنفيذ النص البرمجي. لذلك يجب علينا أن نتحقق من ظهور خطأ لمنع تصرف النص البرمجي بصورة غير مسيطر عليها أو غير محددة بشكل واضح.
مثلاً قد نظن أن جميع الاستعلامات في نص ما قد تم تنفيذها ولكن الواقع قد يكون مختلفاً.

تختلف آلية معالجة الأخطاء بين قواعد البيانات المختلفة وهي موضوع واسع لذلك سنحاول تغطية النقاط الأساسية فيه وترك التفاصيل للبحث من قبل المهتمين.

إعادة قيم بتعبير RETURN:

في قواعد بيانات DB2 وSQL Server يمكن للإجرائية المخزنة إرجاع قيمة باستخدام تعبير RETURN وذلك بالصيغة:

```
RETURN value;
```

عندما يصل التنفيذ إلى حيث يظهر هذا التعبير يتوقف تنفيذ الإجرائية مباشرة.
تستطيع هذه العبارة إعادة قيمة صحيحة فقط.
تستخدم القيمة المعادة لتحديد فيما إذا كان تنفيذ الإجرائية ناجحاً أم لا، عادة تعاد القيمة 0 في حال التنفيذ الناجح ورقم خطأ ما في حال حدوث خلل أثناء التنفيذ.

معالجة الأخطاء

كما ذكرنا تتجاوز الإجرائيات المخزنة حدود الاستعلامات البسيطة وتدخل في إطار لغة برمجة إجرائية.
في مثل هذه البيئة، يعتبر ظهور الأخطاء أحد الاحتمالات الواردة.

قد يتسبب بالخطأ استعلام خاطئ، أو أي عبارة SQL خاطئة، أو قد يكون نتيجة تمرير بيانات غير مناسبة إلى إجرائية مخزنة أو قد يكون أحياناً خطأ مفتعل قمنا بتوليده لسبب ما.

بعكس الأخطاء في بعض لغات البرمجة الأخرى لا تسبب أخطاء SQL بالضرورة بإيقاف تنفيذ النص البرمجي. لذلك يجب علينا أن نتحقق من ظهور خطأ لمنع تصرف النص البرمجي بصورة غير مسيطر عليها أو غير محددة بشكل واضح. مثلاً قد نظن أن جميع الاستعلامات في نص ما قد تم تنفيذها ولكن الواقع قد يكون مختلفاً.

تختلف آلية معالجة الأخطاء بين قواعد البيانات المختلفة وهي موضوع واسع لذلك سنحاول تغطية النقاط الأساسية فيه وترك التفاصيل للبحث من قبل المهتمين.

إعادة قيم بتعبير RETURN:

في قواعد بيانات DB2 وSQL Server يمكن للإجرائية المخزنة إرجاع قيمة باستخدام تعبير RETURN وذلك بالصيغة:
عندما يصل التنفيذ إلى حيث يظهر هذا التعبير يتوقف تنفيذ الإجرائية مباشرة.
تستطيع هذه العبارة إعادة قيمة صحيحة فقط.
تستخدم القيمة المعادة لتحديد فيما إذا كان تنفيذ الإجرائية ناجحاً أم لا، عادة تعاد القيمة 0 في حال التنفيذ الناجح ورقم خطأ ما في حال حدوث خلل أثناء التنفيذ.

معالجة الأخطاء في قواعد بيانات SQL Server

يتوفر في قواعد بيانات SQL Server ما يسمى بالمتحولات العامة التي يمكن الوصول إليها باستخدام @@ قبل اسم المتحول. يعتبر المتحول ERROR أحد أهم هذه المتحولات بالنسبة لموضوع معالجة الأخطاء.

يأخذ هذا المتحول القيمة 0 في حال عدم ظهور أي خطأ وإلا فإنه يأخذ قيمة عددية صحيحة تعتمد على نوع الخطأ الذي ظهر. يمكننا الاستفادة من هذا التعبير باستخدامه كشرط لتعبير IF للتحقق من ظهور خطأ وذلك بالصيغة:

```
IF (@@ERROR <> 0)
errorHandler;
```

حيث تمثل errorHandler النص البرمجي الذي سوف يقوم بمعالجة الخطأ الحاصل.

قد يكون أحد السيناريوهات المحتملة الرغبة في إيقاف التنفيذ أو حفظ الخطأ في متحول خرج للتدقيق فيه لاحقاً.

قد يلزمنا أحياناً أن نولد خطأ ما ولذلك يمكننا استخدام التعبير RAISERROR وذلك بالصيغة:

```
RAISERROR {msg_id | msg_str}, severity, state [,argument];
```

حيث تمثل msg_id رقم الخطأ حيث يجب أن يكون بين 13000 و2147483647.
وتمثل msg_str رسالة الخطأ التي يجب أن تكون معرفة في جدول رسائل النظام.
وseverity هي قيمة تتراوح بين 0 و25 تمثل خطورة الخطأ حيث الأخطاء ذات الشدة التي تتجاوز القيمة 20 هي أخطاء ستؤدي إلى فصل الاتصال.
تمثل state قيمة بين 1 حتى 127 لتعيد معلومات عن حالة الخطأ.

يمكننا أيضاً تمرير معاملات إضافية إلى نص الرسالة. في هذه الحالة يجب أن يحتوي نص الرسالة إشارة تحدد مكان توضع قيمة هذا المعامل ضمن الرسالة. تتكون هذه الإشارة من إشارة % مع محرف لتحديد نوع المعامل. فعلى سبيل المثال، يمثل d% عدد صحيح سالب أو موجب بينما يمثل u% عدد صحيح موجب وتمثل s% سلسلة محارف.

معالجة الأخطاء في قواعد بيانات SQL Server

يتوفر في قواعد بيانات SQL Server ما يسمى بالمتحولات العامة التي يمكن الوصول إليها باستخدام @@ قبل اسم المتحول.
يعتبر المتحول ERROR أحد أهم هذه المتحولات بالنسبة لموضوع معالجة الأخطاء.

يأخذ هذا المتحول القيمة 0 في حال عدم ظهور أي خطأ وإلا فإنه يأخذ قيمة عددية صحيحة تعتمد على نوع الخطأ الذي ظهر.
يمكننا الاستفادة من هذا التعبير باستخدامه كشرط لتعبير IF للتحقق من ظهور خطأ
قد يكون أحد السيناريوهات المحتملة الرغبة في إيقاف التنفيذ أو حفظ الخطأ في متحول خرج للتدقيق فيه لاحقاً.

قد يلزمنا أحياناً أن نولد خطأ ما ولذلك يمكننا استخدام التعبير RAISERROR.

يمكننا أيضاً تمرير معاملات إضافية إلى نص الرسالة. في هذه الحالة يجب أن يحتوي نص الرسالة إشارة تحدد مكان توضع قيمة هذا المعامل ضمن الرسالة. تتكون هذه الإشارة من إشارة % مع محرف لتحديد نوع المعامل. فعلى سبيل المثال، يمثل d% عدد صحيح سالب أو موجب بينما يمثل u% عدد صحيح موجب وتمثل s% سلسلة محارف.

مثال:

إذا أردنا كتابة الإجرائية المخزنة التي تقوم بإدراج سجل ضمن جدول Customers مع التأكد من عدم تكرار الرقم التسلسلي للزبون نكتب الصيغة:

```
CREATE PROCEDURE safeInsert
(@myCustomerID INT, @myCustomerName varchar(50))
AS
BEGIN
DECLARE @theError INT;
Insert Into Customers Values (@myCustomerID, @myCustomerName);
SET @theError = @@ERROR;
IF @theError <> 0
BEGIN
RAISERROR
('cannot insert the customer with ID %d',
10, 1, @customerID);
RETURN @theError;
END;
ELSE
RETURN 0;
END;
```

نلاحظ في هذا المثال أننا:

- قمنا بإنشاء الإجرائية safeInsert
- ثم صرحنا عن متحول theError ليحمل قيمة الخطأ التي يعيدها المتحول العام @@ERROR والتي تولدت عن تنفيذ الاستعلام الذي سيضيف سجل إلى الجدول Customers.
- تحققنا من قيمة theError
    - فإذا كانت مساوية للصفر نعتبر أن العملية قد تمت بنجاح وبالتالي ستعاد القيمة 0 من الإجرائية.
    - أما إذا كانت قيمة theError لاتساوي الصفر فهذا دليل على أن خللاً ما قد حصل. لذا قمنا بتوليد خطأ، يظهر في رسالته رقم السجل المضاف، ورقم الزبون الذي يحلّ في الرسالة محلّ الإشارة d%.

الآن يمكننا تنفيذ هذه الإجرائية بالصيغة:

```
EXEC safeInsert 20, 'Adel';
```

معالجة الأخطاء في قواعد بيانات Oracle

تختلف الأمور قليلاً في قواعد بيانات Oracle حيث يتم استخدام كتلة مرافقة لتعبير EXCEPTION في نهاية رماز الإجرائية المخزنة، وذلك قبل التعبير END، حيث تتم كتابة النص البرمجي الذي سينفذ عند حدوث خطأ ما ضمن هذه الكتلة.

يمكن إظهار التركيب كاملاً بالصيغة:

```
BEGIN
--SQL Code
EXCEPTION
WHEN Exception1 THEN
--handelException1
WHEN EXCEPTION2 THEN
--handelException2
END;
```

يوجد مجموعة من الاستثناءات المعرفة مسبقاً كما يمكننا تعريف استثناءات جديدة وكتابة نصوص برمجية خاصة بها.

أكثر الاستثناءات المعرفة مسبقاً شهرة هي التالية:

* **CURSOR_ALREADY_OPEN:** يظهر هذا الخطأ إذا حاولنا فتح مؤشر تم فتحه مسبقاً.
* **DUP_VAL_ON_INDEX:** يظهر هذا الخطأ إذا حاولنا إدراج سجل بقيمة مكررة للمفتاح الرئيسي أو لحقل مقيد بقيد فريد.
* **INVALID_NUMBER:** يظهر هذا الخطأ إذا حاولنا تحويل سلسلة محارف إلى رقم إذا كانت سلسلة المحارف تلك لاتحتوي قيمة رقمية صالحة.
* **NO_DATA_FOUND:** يظهر هذا الخطأ إذا استخدمنا التعبير SELECT INTO لتخزين قيمة في متحول ولكن لم تتم إعادة أي قيمة من هذا الاستعلام.
* **TO_MANY_ROWS:** يظهر هذا الخطأ إذا استخدمنا التعبير SELECT INTO لتخزين قيمة في متحول وأعاد الاستعلام أكثر من سجل.
* **OTHERS:** يحدد جميع الاستثناءات الغير مشمولة بما سبق.

لتعريف استثناء جديد يكفي التصريح عن متغير من النوع EXCEPTION وإطلاق هذا الاستثناء في حال تحقق شرط ما حيث تتم معالجة هذا الاستثناء بالصورة نفسها التي تتم فيها معالجة الاستثناءات مسبقة التعريف.

معالجة الأخطاء في قواعد بيانات Oracle

تختلف الأمور قليلاً في قواعد بيانات Oracle حيث يتم استخدام كتلة مرافقة لتعبير EXCEPTION في نهاية رماز الإجرائية المخزنة، وذلك قبل التعبير END، حيث تتم كتابة النص البرمجي الذي سينفذ عند حدوث خطأ ما ضمن هذه الكتلة.

يوجد مجموعة من الاستثناءات المعرفة مسبقاً كما يمكننا تعريف استثناءات جديدة وكتابة نصوص برمجية خاصة بها.

أكثر الاستثناءات المعرفة مسبقاً شهرة هي التالية:

* **CURSOR_ALREADY_OPEN:** يظهر هذا الخطأ إذا حاولنا فتح مؤشر تم فتحه مسبقاً.
* **DUP_VAL_ON_INDEX:** يظهر هذا الخطأ إذا حاولنا إدراج سجل بقيمة مكررة للمفتاح الرئيسي أو لحقل مقيد بقيد فريد.
* **INVALID_NUMBER:** يظهر هذا الخطأ إذا حاولنا تحويل سلسلة محارف إلى رقم إذا كانت سلسلة المحارف تلك لاتحتوي قيمة رقمية صالحة.
* **NO_DATA_FOUND:** يظهر هذا الخطأ إذا استخدمنا التعبير SELECT INTO لتخزين قيمة في متحول ولكن لم تتم إعادة أي قيمة من هذا الاستعلام.
* **TO_MANY_ROWS:** يظهر هذا الخطأ إذا استخدمنا التعبير SELECT INTO لتخزين قيمة في متحول وأعاد الاستعلام أكثر من سجل.
* **OTHERS:** يحدد جميع الاستثناءات الغير مشمولة بما سبق.

لتعريف استثناء جديد يكفي التصريح عن متغير من النوع EXCEPTION وإطلاق هذا الاستثناء في حال تحقق شرط ما حيث تتم معالجة هذا الاستثناء بالصورة نفسها التي تتم فيها معالجة الاستثناءات مسبقة التعريف.

معالجة الأخطاء في قواعد بيانات Oracle

مثال:

إذا أردنا إنشاء الإجرائية المخزنة التي تقوم بإدراج اسم ورقم طالب ضمن الجدول Students مع التحقق بأن اسم الطالب ليس 'Sami' وأن رقمه التسلسلي الذي يشكل مفتاح رئيسي للجدول Students غير مكرر.

```
CREATE OR REPLACE PROCEDURE insertSudent( myStudentName IN
varchar(50),myStudentID IN INT)
AS
myCustomException EXCEPTION;
BEGIN
      IF myStudentName =’sami’THEN
            RAISE myCustomException ;
      END IF;
      Insert into students values (myStudentID,myStudentName);
EXCEPTION
WHEN DUP_VAL_ON_INDEX THEN
Dbms_output.put_line(‘we have this ID ion our table’);
WHEN myCustomException THEN
Dbms_output.put_line(‘this student is not allowed to register’);
END;
```

نلاحظ في المثال السابق أننا قمنا بإدراج سجل ضمن الجدول Students، وقمنا بمعالجة استثنائين: الأول مسبق التعريف هو DUP_VAL_ON_INDEX، والثاني هو myCustomException الذي قمنا بالتصريح عنه في بداية الإجرائية. ثم قمنا بكتابة كتلة المعالجة بعد التعبير EXCEPTION حيث ستتم طباعة الرسالة الأولى في حالة الاستثناء الأول والثانية في حالة الاستثناء الثاني.

لاستخدام هذه الإجرائية نكتب الصيغة:

```
CALL insertStudent (‘Adel’, 10);
```

معالجة الأخطاء في قواعد بيانات DB2

تعتمد قواعد بيانات DB2 على متحولين لمعالجة الأخطاء في الإجرائيات المخزنة. هذين المتحولين هما:

- SQLSTATE: الذي يتكون من خمس محارف، ويمثل الرمز المعياري لحالة الخطأ.
- SQLCODE: وهو عبارة عن متحول صحيح.

نستطيع استخدام أحد هذين المتحولين وليس كليهما لأن تعيين أحدهما سوف يلغي الآخر. على كل حال، نحتاج أن نصرح عن هذين المتحولين قبل استخدامهما وذلك بالصيغة:

```
DECLARE SQLCODE INT DEFAULT 0;
```

أو في حالة SQLSTATE بالشكل:

```
DECLARE SQLSTATE CHAR(5) DEFAULT ‘00000’;
```

سيسبب ظهور أي خطأ أثناء تنفيذ الإجرائية المخزنة، في إيقاف عملية التنفيذ. لذا يجب أن يكون لدينا معالج أخطاء يحدد ما الذي سيحصل في حال ظهور الخطأ.

لنتمكن من التحكم بالتنفيذ في حال ظهور خطأ سنقوم بإعداد نص برمجي لتلقي أي استثناء وذلك من خلال التعبير DECLARE…HANDLER حيث نكتب الصيغة:

```
DECLARE handler_type HANDLER FOR error_type
BEGIN
--handler_code
END;
```

حيث تمثل handler_type نوع معالج الأخطاء وهو واحد من الخيارات التالية:

- **CONTINUE:** يحدد هذا النوع أن النص ضمن معالج الأخطاء سوف يتم تنفيذه ثم سيتابع التنفيذ من السطر التالي تماماً إلى النقطة التي ظهر فيها الخطأ.
- **EXIT:** ويستخدم عندما نريد أن تتم متابعة التنفيذ بعد تنفيذ نص المعالج بعد كتلة BEGIN…END التي ظهر فيها الخطأ وإذا حصل الخطأ في كتلتي BEGIN…END متداخلتين سيتابع التنفيذ للنص البرمجي في الكتلة الخارجية إذا حصل الخطأ في الكتلة الداخلية.

أما بالنسبة لـ error_type فتحدد نوع الخطأ المرتبط بمعالج الأخطاء هذا.

يمكننا معالجة أخطاء محددة أو أحد الحالات العامة التالية:

- SQLEXCEPTION: سيتم تشغيل النص البرمجي في معالج الأخطاء عند ظهور أي استثناء SQL.
- SQLWARNING: يتم تشغيل النص البرمجي الخاص بالمعالج عند ظهور أي تحذير SQL.
- NOT FOUND: سيتم تنفيذ النص البرمجي الخاص بالمعالج عندما لا يعيد التعبير WHERE الخاص باستعلام أي سجل مطابق.

معالجة الأخطاء في قواعد بيانات DB2

تعتمد قواعد بيانات DB2 على متحولين لمعالجة الأخطاء في الإجرائيات المخزنة. هذين المتحولين هما:

- SQLSTATE: الذي يتكون من خمس محارف، ويمثل الرمز المعياري لحالة الخطأ.
- SQLCODE: وهو عبارة عن متحول صحيح.

نستطيع استخدام أحد هذين المتحولين وليس كليهما لأن تعيين أحدهما سوف يلغي الآخر.على كل حال، نحتاج أن نصرح عن هذين المتحولين قبل استخدامهما.

سيسبب ظهور أي خطأ أثناء تنفيذ الإجرائية المخزنة، في إيقاف عملية التنفيذ. لذا يجب أن يكون لدينا معالج أخطاء يحدد ما الذي سيحصل في حال ظهور الخطأ.

لنتمكن من التحكم بالتنفيذ في حال ظهور خطأ سنقوم بإعداد نص برمجي لتلقي أي استثناء قد يحصل وذلك من خلال التعبير DECLARE…HANDLER.

حيث تمثل handler_type نوع معالج الأخطاء وهو واحد من الخيارات التالية:

- **CONTINUE:** يحدد هذا النوع أن النص ضمن معالج الأخطاء سوف يتم تنفيذه ثم سيتابع التنفيذ من السطر التالي تماماً إلى النقطة التي ظهر فيها الخطأ.
- **EXSIT:** ويستخدم عندما نريد أن تتم متابعة التنفيذ بعد تنفيذ نص المعالج بعد كتلة BEGIN…END التي ظهر فيها الخطأ وإذا حصل الخطأ في كتلتي BEGIN…END متداخلتين سيتابع التنفيذ للنص البرمجي في الكتلة الخارجية إذا حصل الخطأ في الكتلة الداخلية.

مثال:

إذا أردنا معالجة الحالة الخاصة بـ SQL المحددة بالمحارف الخمسة '23405' والتي تظهر عندما نحاول إدراج قيمة مكررة في حقل مفتاح رئيسي، أو في حقل مقيد بقيد فريد، نكتب الصيغة التالية التي تسمح لنا بمتابعة التنفيذ عند العبارة التالية للعبارة التي أطلقت الاستثناء:

```
DECLARE CONTINUE HANDLER FOR SQLSTATE ‘23505’
--code to handle the error;
```

تُستخدم الصيغة السابقة في حال أردنا التعامل مع استثناءات مسبقة الإعداد. أما إنشاء استثناءات جديدة فيحتاج لاستخدام التعبير SIGNAL وذلك بالصيغة:

```
SIGNAL SQLSTATE SQLStateCode
SET MESSAGE_TEXT = errorDescription;
```

تطلق الصيغة السابقة استثناء باستخدام متحول الحالة SQLSTATE، والذي سوف يتم تمريره إلى التطبيق الذي استدعى الإجرائية، في حال لم تتم معالجته من قبل معالج الأخطاء.

انتبه:

لا يمكنك تحديد قيمة المحارف في متحول الحالة خارج المجال من 7 إلى 9 أو من T إلى Z بالنسبة للاستثناءات التي يتم إنشاؤها لأن القيم خارج هذا المجال محجوزة للاستثناءات مسبقة الإعداد.

معالجة الأخطاء في قواعد بيانات DB2

مثال:

إذا أردنا استخدام متحول الحالة SQLSTATE في كتابة صيغة الإجرائية المخزنة التي تبحث عن اسم ضمن جدول أسماء، وتعيد رسالة بوجوده إذا كان موجوداً، أو بعدم وجوده إذا كان غير متوفراً، نكتب الصيغة:

```
CREATE  searchFor (myName varchar(50), OUT myMessage varchar(50))
P1: BEGIN
DECLARE SQLSTATE char(5) DEFAULT ‘00000’;
DECLARE EXIT HANDLER FOR NOT FOUND
myMessage = ‘did not find name’;
IF myName = ‘sami’ THEN
SIGNAL SQLSTATE ‘87000’
SET MESSAGE_TEXT = ‘can not search for that name’;
myMessage = ‘can not search for that name’;
END IF;
Select name from Names where name = myName;
END p1;
```

في هذا المثال نلاحظ أننا قمنا بالتصريح عن المتحول SQLSTATE، ثم صرحنا عن معالج استثناءات من النوع EXIT الذي يعالج الخطأ من النوع NOT FOUND، ثم كتبنا الشرط الذي يحدد كون الاسم المُدخل مساوي لقيمة معينة، وأطلقنا استثناء برقم الحالة ‘87000’.

لاستخدام هذه الإجرائية نقوم بكتابة الصيغة:

```
CALL searchFor (‘samer’ ,?);
```

معالجة الأخطاء في قواعد بيانات MySQL

تشابه الطريقة التي تتعامل بها MySQL مع الأخطاء، الطريقة المستخدمة في DB2.

يبدأ العمل بالتصريح عن معالج لتولي خطأ أو تحذير أو استثناء باستخدام الصيغة التالية:

```
DECLARE handler_type HANDLER FOR condition_value  [,...] sp_statement;
```

بعد حدوث الخطأ عن:

- CONTINUE: سيستمر التنفيذ من العبارة التالية للعبارة التي سببت الخطأ
- EXIT: سيتم الخروج لتنفيذ أول عبارة خارج الكتلة END-BEGIN التي سببت الخطأ

تتوزع قيم الأخطاء condition_value في MYSQL على مجالات رقمية يمكن تصنيفها كما يلي:

- SQLWARNING تبدأ القيم ب 01
- NOT FOUND تبدأ القيم ب 02
- SQLEXCEPTION أي قيمة خارج القيم السابقة.
- mysql_error_code: تستخدم لمعالجة الأخطاء بصورة إفرادية. فعلى سبيل المثال، قد نرغب في معالجة خطأين من النوع NOT FOUND بصورة مختلفة باستخدام رمز خطأ محدد في كل منهما.
- Condition_name: يسمح لنا بتحديد أسماء خاصة بنا لرمز خطأ ما.
- Sp_statement: تعبر عن العبارة البرمجية التي سيتم تنفيذها في حال ظهور خطأ. غالباً ما نستخدم العبارة SET لإعطاء قيمة إلى متحول بحيث يشير هذا المتحول إلى الخطأ الحاصل.

تشابه الطريقة التي تتعامل بها MySQL مع الأخطاء، الطريقة المستخدمة في DB2.

يبدأ العمل بالتصريح عن معالج لتولي خطأ أو تحذير أو استثناء

بعد حدوث الخطأ عن:

- CONTINUE: سيستمر التنفيذ من العبارة التالية للعبارة التي سببت الخطأ
- EXIT: سيتم الخروج لتنفيذ أول عبارة خارج الكتلة END-BEGIN التي سببت الخطأ

تتوزع قيم الأخطاء condition_value في MYSQL على مجالات رقمية يمكن تصنيفها كما يلي:

- SQLWARNING تبدأ القيم ب 01
- NOT FOUND تبدأ القيم ب 02
- SQLEXCEPTION أي قيمة خارج القيم السابقة.
- mysql_error_code: تستخدم لمعالجة الأخطاء بصورة إفرادية. فعلى سبيل المثال، قد نرغب في معالجة خطأين من النوع NOT FOUND بصورة مختلفة باستخدام رمز خطأ محدد في كل منهما.
- Condition_name: يسمح لنا بتحديد أسماء خاصة بنا لرمز خطأ ما.
- Sp_statement: تعبر عن العبارة البرمجية التي سيتم تنفيذها في حال ظهور خطأ. غالباً ما نستخدم العبارة SET لإعطاء قيمة إلى متحول بحيث يشير هذا المتحول إلى الخطأ الحاصل.

معالجة الأخطاء في قواعد بيانات MySQL

مثال1:

إذا أردنا كتابة الإجرائية المخزنة التي تقوم بإدراج قيم معينة ضمن الجدول emps والتحسس للأخطاء التي قد تظهر نكتب الصيغة:

```
CREATE PROCEDURE handlerproc(OUT p_end VARCHAR(10))
BEGIN
  declare continue handler for 1062 SET @b = '- With Error 1062';
  declare continue handler for 1048 SET @b = '- With Error 1048';

  insert into emps VALUES (NULL,'Dave',1,10) ;

  set p_end:= concat('The End ',@b);

END;
```

نلاحظ هنا أننا قمنا بالتصريح عن معالجي أخطاء:

- أحدهما لقيمة رمز الخطأ 1062 ER_DUP_ENTRY التي تشير إلى خطأ إدخال قيمة مكررة في حقل إدخال
- والآخر لرمز الخطأ 1048 ER_BAD_NULL_ERROR الذي يظهر عند إدخال قيمة NULL في حقل لا يقبل هذه القيمة.

نلاحظ أننا مررنا قيمة المتحول b إلى رسالة سيتم إظهارها عند انتهاء تنفيذ العبارات كلها كوننا استخدمنا Continue كخيار لمعالجي الأخطاء.

مثال 2:

نستخدم في هذا المثال تصريح عن شرط ما مقابل لقيمة SQLSTATE التي تساوي ' 23000' واستخدام اسم هذا الشرط كبديل عن قيمة SQLSTATE

```
create procedure conditionproc(OUT p_end VARCHAR(10))
begin

  declare not_null condition for SQLSTATE '23000';
  declare continue handler for not_null SET @b = '- With not_null
Error';

  insert into emps VALUES (NULL,'Dave',1,10) ;

  set p_end:= concat('The End ',@b);

end;
```

حول الإجرائيات المخزنة

رأينا من خلال الجلسات السابقة قوة الإجرائيات المخزنة وكيف أنها تتضمن آليات للتنفيذ الشرطي والحلقات والمؤشرات.

ترى بعض النظريات ضرورة التركيز على تنفيذ جميع الاستعلامات عن طريق الإجرائيات المخزنة وذلك للأسباب التالية:

1- غالباً ما تتم عملية ترجمة مسبقة للإجرائيات المخزنة في قواعد البيانات العلائقية مما يجعل استخدامها فرصة لتحسين الأداء.
2- تعتبر الإجرائيات المخزنة وسيلة جيدة لرفع درجة الأمان كونها توقف مجموعة من الهجومات الأمنية المحتملة على قاعدة البيانات.
3- تضمين منطق العمل في قاعدة البيانات نفسها وجعل التطبيقات تلعب دور واجهة تعامل مع البيانات فقط.

القسم الثاني عشر

إدارة الأمن في قواعد البيانات باستخدام SQL

الكلمات المفتاحية:

الأمن ، التحقق من الهوية ، الصلاحيات ، حساب ، مستخدم، دور.

ملخص:

يعتبر الأمن في قواعد البيانات موضوعاً واسعاً جداً. سنحاول في هذا القسم تغطية الأفكار الرئيسية المتعلقة بتعليمات SQL الخاصة بأمن قاعدة بيانات، كما سنستعرض أسلوب تطبيقها في قواعد البيانات المختلفة.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- مفهوم الأمن بصورة عامة.
- التعليمات الأساسية المرتبطة بإدارة السماحيات التي يحددها المعيار SQL99
- كيفية إدارة الأمن في ACCESS
- كيفية إدارة الأمن في SQL SERVER
- كيفية إدارة المستخدمين فقط في ORACLE (سنتابع الأفكار المتبقية في القسم التالي)

SQL وأمن البيانات

كلما ازدادت أهمية البيانات وحساسيتها بالنسبة للعمل، زادت ضرورة الحفاظ عليها بمنأى عن الأيدي العابثة.
ولا تقتصر عملية ضبط الأمان على الحماية من محاولات اختراق خارجية، بل تتعداها إلى تنظيم الوصول إلى البيانات من قبل المستخدمين المختلفين، مما يستدعي مثلاً حجب جدول الرواتب عن الموظفين خارج قسم المحاسبة.
لن نتناول في هذه الجلسة نظرية الأمن ولكننا سنركز فقط على أوامر SQL التي تمكن من إدارة وضبط الأمان.
عند تطبيق نظامٍ أمني لابد لنا من التفكير بإجرائيتين أساسيتين:

1- التحقق من الهوية: حيث يتم التأكد من هوية المستخدم الذي يحاول الولوج إلى القاعدة. تتم هذه العملية باستخدام اسم دخول وكلمة مرور (أو كلمة سر).
2- الصلاحيات والسماحيات: يجب تحديد الصلاحيات الخاصة بالمستخدم أي ما يسمح لهذا المستخدم فعله بعد التحقق من هويته، وتُحدد السماحيات الخاصة بكل غرض من أغراض القاعدة، أي ما يمكن فعله بهذا الغرض.

تمنح أنظمة قواعد البيانات العلائقية العديد من الأدوات التي تساعد على ضبط عملية الأمن وإجرائياتها.
سنوجه اهتمامنا إلى التعامل مع حاجات الأمن الأساسية التي تتشابه فيها طرق التطبيق بين أنظمة قواعد البيانات المختلفة.

لمحة عامة:

عند تثبيت قاعدة بيانات معينة يتم إنشاء حسابات تلقائية بصلاحيات مدير نظام. لكن، لإعطاء مستخدم ما صلاحية الوصول إلى القاعدة، علينا أن نمر بمرحلتين:

1- إنشاء حساب لهذا المستخدم إذا كان لا يملك حساباً مسبقاً: بهذه الطريقة يستطيع نظام قواعد البيانات التحقق من هوية هذا الشخص. بصورة عامة يكون حساب هذا المستخدم محمياً بكلمة سر يُدخلها المستخدم أثناء عملية تسجيل الدخول.
2- عند إنشاء حساب مستخدم، لن يكون لهذا المستخدم أي صلاحيات للوصول إلى أي غرض من أغراض قاعدة البيانات.
لذلك يكون على مدير القاعدة توزيع صلاحيات محددة على المستخدم تعطيه إمكانية الوصول إلى الأغراض التي يحتاجها في قاعدة البيانات.

يمكن توزيع صلاحيات مختلفة، مثل صلاحية إنشاء جداول جديدة، أو تعديل سجلات، أو استعلام عن سجلات، أو إدراج سجلات أو حذف سجلات من جدول ما. كما يمكن للمستخدم الحصول على صلاحية تنفيذ إجرائيات مُخزنة، أو حتى صلاحية التعديل على حسابات قواعد البيانات المختلفة.
قد تكون الصلاحيات المعطاة شاملة لكافة الجداول، أو خاصة بجدول محدد.
تدعم قواعد البيانات عدد كبير من الصلاحيات الممكنة للمستخدم، إلا أن صلاحيات مدير النظام تبقى الصلاحيات الأقوى التي تساعد على تحديد، من الذي وما الذي يستطيع المستخدم فعله في قاعدة البيانات.

الأدوار والمناظير في قاعدة البيانات

الأدوار:

- تبنت SQL حلولاً لتوزيع صلاحيات كثيرة عبر دعم ما يُسمى بالأدوار.
- يُعرّف الدور بأنه تسميةٌ لمجموعة من الصلاحيات.
- يستطيع مدير النظام أن يعطي للمستخدم دور عوضاً عن إعطائه عشرات الصلاحيات.
- لاتدعم قواعد البيانات MySQL الأدوار، في حين تدعم Access ما يسمى بمجموعة مستخدمين.
- تحتوي قواعد البيانات التي تدعم الأدوار عادةً، على أدوار مسبقة الإعداد كما هو الحال في دور sysAdmin المعرّف مسبقاً في SQL Server.

المناظير:

- استخدام المناظير كآلية لحجب بعض المعلومات من الجداول
- وإعطاء المستخدم صلاحية التعامل مع منظار للتعامل مع الحقول المسموح له رؤيتها فقط.

الأدوار:

مع كل المرونة التي يمنحها تزويد المستخدم بأية تركيبة من الصلاحيات، تصبح هذه العملية مرهقة وإشكالية في حال وجود عدد كبير من المستخدمين. لذا تبنت SQL حلولاً لمثل هذه المشكلة عبر دعم ما يُسمى بالأدوار.

يُعرّف الدور بأنه تسميةٌ لمجموعة من الصلاحيات. فعند إسناد دور إلى مستخدم ما نحصل على نفس التأثير الذي ينتج عن إسناد جميع الصلاحيات المُعرفة ضمن هذا الدور إلى المستخدم.

للأدوار أهمية كبيرة في تسهيل عملية إدارة الأمن في قاعدة البيانات، إذ يستطيع مدير النظام أن يعطي للمستخدم دور عوضاً عن إعطائه عشرات الصلاحيات.

لاتدعم قواعد البيانات MySQL الأدوار، في حين تدعم Access ما يسمى بمجموعة مستخدمين التي يقترب مفهومها العام من مفهوم الأدوار.

تحتوي قواعد البيانات التي تدعم الأدوار عادةً، على أدوار مسبقة الإعداد كما هو الحال في دور sysAdmin المعرّف مسبقاً في SQL Server والذي ينتمي إليه الحساب sa، حيث يكفي إسناد هذا الدور إلى حساب ما لإعطاءه صلاحية مطلقة.

المناظير:

تكلمنا عن المناظير ولم نحدد ارتباطها بموضوع الأمن. فمن المألوف بالنسبة لمدير نظام أن يستخدم المناظير كآلية لحجب بعض المعلومات من الجداول، وإعطاء المستخدم صلاحية التعامل مع منظار للتعامل مع الحقول المسموح له رؤيتها فقط عوضاً عن إعطاء الصلاحية على الجداول كاملةً.

أمن قواعد البيانات وSQL

لا يحتوي معيار SQL99 أي أوامر للتحقق من الهوية لذا سنستعرض الاختلافات بين طريقة تطبيق كل قاعدة بيانات لهذا الموضوع.

أما بالنسبة للصلاحيات، فيحدد المعيار SQL99 أمرين أساسيين لإدارتها:GRANT وREVOKE حيث تدعم معظم قواعد البيانات هذين الأمرين بالرغم من اختلاف الصيغة بشكل بسيط بينها.

كذلك يحتوي المعيار SQL99 الأوامر الخاصة بإنشاء وإدارة الأدوار وهي CREATE ROLE وSET ROLE و DROP ROLE.

استخدام التعبير GRANT:

يتم استخدام التعبير GRANT لتخويل مستخدم بصلاحية ما. هناك نوعان من صلاحيات:

1- صلاحيات مستوى التصريح: وهي الصلاحيات التي تتضمن عمليات غير مرتبطة بأغراض موجودة محددة في قاعدة البيانات مثل CREATE DATABASE, CREATE TABLE, CREATE PROCEDURE, CREATE VIEW. لتخويل مستخدم بصلاحية التصريح يكفي كتابة الصيغة:

```
GRANT prevelage_type TO user_name
```

2- صلاحيات على مستوى الأغراض: تتحكم هذه الصلاحيات بالوصول إلى أغراض مخصصة موجودة في قاعدة البيانات مثل تحديد صلاحية الوصول إلى الجداول حيث يمكن أن تكون INSERT, UPDATE, DELETE, SELECT أو تركيبة من هذه الصلاحيات. أما بالنسبة للصلاحيات على الإجرائيات المخزنة فأهم الصلاحيات هي EXECUTE.

كما سنلاحظ ففي حالة الصلاحيات على مستوى الأغراض يجب علينا تحديد المصدر إضافة إلى نوع الصلاحية واسم المستخدم. تصبح الصيغة بالشكل:

```
GRANT privilege_type ON resource TO user_name
```

يملك المستخدمون صلاحيات كاملة على أغراض قاعدة البيانات التي يقومون بإنشائها بينما لا يملكون أي صلاحية على الأغراض المنشأة من قبل مستخدمين آخرين.

في قواعد بيانات SQL Server، Oracle وDB2 يمكنك استخدام التعبير GRANT لتخصيص أدوار خاصة بمستخدمين موجودين.

أمن قواعد البيانات وSQL

لا يحتوي معيار SQL99 أي أوامر للتحقق من الهوية لذا سنستعرض الاختلافات بين طريقة تطبيق كل قاعدة بيانات لهذا الموضوع.

أما بالنسبة للصلاحيات، فيحدد المعيار SQL99 أمرين أساسيين لإدارتها:GRANT وREVOKE حيث تدعم معظم قواعد البيانات هذين الأمرين بالرغم من اختلاف الصيغة بشكل بسيط بينها.

كذلك يحتوي المعيار SQL99 الأوامر الخاصة بإنشاء وإدارة الأدوار وهي CREATE ROLE وSET ROLE و DROP ROLE.

استخدام التعبير GRANT:

يتم استخدام التعبير GRANT لتخويل مستخدم بصلاحية ما. هناك نوعان من صلاحيات:

1- صلاحيات مستوى التصريح: وهي الصلاحيات التي تتضمن عمليات غير مرتبطة بأغراض موجودة محددة في قاعدة البيانات مثل CREATE DATABASE, CREATE TABLE, CREATE PROCEDURE, CREATE VIEW.

2- صلاحيات على مستوى الأغراض: تتحكم هذه الصلاحيات بالوصول إلى أغراض مخصصة موجودة في قاعدة البيانات مثل تحديد صلاحية الوصول إلى الجداول حيث يمكن أن تكون INSERT, UPDATE, DELETE, SELECT أو تركيبة من هذه الصلاحيات. أما بالنسبة للصلاحيات على الإجرائيات المخزنة فأهم الصلاحيات هي EXECUTE.

كما سنلاحظ ففي حالة الصلاحيات على مستوى الأغراض يجب علينا تحديد المصدر إضافة إلى نوع الصلاحية واسم المستخدم.

يملك المستخدمون صلاحيات كاملة على أغراض قاعدة البيانات التي يقومون بإنشائها بينما لا يملكون أي صلاحية على الأغراض المنشأة من قبل مستخدمين آخرين.
في قواعد بيانات SQL Server، Oracle وDB2 يمكنك استخدام التعبير GRANT لتخصيص أدوار خاصة بمستخدمين موجودين.

استخدام التعبير REVOKE:

يستخدم التعبير REVOKE لإزالة أو حجب صلاحيات تم إعطاؤها بالتعبير GRANT أو حصل عليها المستخدم بشكل تلقائي.

تعمل REVOKE كما هي الحالة بالنسبة لـ GRANT على سويتين:

- الأولى هي سوية التصريح ويمكن التعامل معها بالصيغة:

```
REVOKE prevelage_type FROM user_name
```

- الثانية هي سوية الأغراض وهنا يلزم تحديد المصدر أي تصبح الصيغة على الشكل:

```
REVOKE prevelage_type ON resource FROM user_name
```

عندما يتم حجب جميع صلاحيات مستخدم، فإن هذا لا يعني أننا حذفنا المستخدم أو أي من الأغراض التي أنشأها وإنما نكون ببساطة قد منعناه من الوصول إلى أي مصدر.

الشكل الكامل للصيغ التي يحددها المعيار SQL99 للتعبير REVOKE وGRANT هي التالية:

REVOKE	GRANT
REVOKE [GRANT OPTION FOR] {ALL PRIVILEGES} | SELECT | INSERT | DELETE | UPDATE | REFERENCES | USAGE }[,…n] ON { [TABLE] table_name | DOMAIN domain_name | COLLATION collation_name | CHARACTER SET charset_name | TRANSLATION translation_name FROM (grantee_name | PUBLIC} [,…n] {CASCAD | RESTRICT}	GRANT {ALL PRIVILEGES} |SELECT |INSERT [(column_name [,…n] |DELETE |UPDATE [(column_name [,…n] |REFERENCES [(column_name [,…n] |USAGE } [,…n] ON {[TABLE] table_name |DOMAIN domain_name |COLLATION collation_name

لن ندخل في تفاصيل معاملات هذه الصيغة العامة حيث ستتوضح بعض هذه المعاملات من خلال الأمثلة التطبيقية في قواعد البيانات المختلفة.

استخدام التعبير REVOKE:

يستخدم التعبير REVOKE لإزالة أو حجب صلاحيات تم إعطاؤها بالتعبير GRANT أو حصل عليها المستخدم بشكل تلقائي.

تعمل REVOKE كما هي الحالة بالنسبة لـ GRANT على سويتين:
الأولى هي سوية التصريح، والثانية هي سوية الأغراض وهنا يلزم كما ذكرنا تحديد المصدر.

عندما يتم حجب جميع صلاحيات مستخدم، فإن هذا لا يعني أننا حذفنا المستخدم أو أي من الأغراض التي أنشأها وإنما نكزن ببساطة قد منعناه من الوصول إلى أي مصدر.

أمن قواعد البيانات وSQL

الأمن في قواعد بيانات MS Access:
يعتبر الأمن في نظام إدارة قواعد المعطيات Access بسيطاً.

يدعم Access 2002 تعابير GRANT وREVOKE. يمكننا الوصول إلى خيارات الأمان في Access من قائمة Tools، ومن خيار Security.

تتلخص الطريقة الأساسية لتأمين قاعدة بيانات Access في وضع كلمة سر عليها من قائمة Tools، الخيار Security، والأمر set database password.

يمكن استخدام تشفير قاعدة البيانات من الأمر Encrypt/Decrypt Database من القائمة Tools، والخيار Security.

يدعم Access الأمن على مستوى المستخدمين ويمتلك واجهة خاصة لإدارة المستخدمين يمكن الوصول إليها من الخيار User and Group Accounts من القائمة Tools، والخيار Security بشكل يمكّن المدير من إزالة وإضافة مستخدمين.

بعد إضافة المستخدمين يمكننا التحكم بصلاحياتهم باستخدام الخيار User and Group Permissions.

أمن قواعد البيانات وSQL

الأمن في قواعد بيانات MS Access:

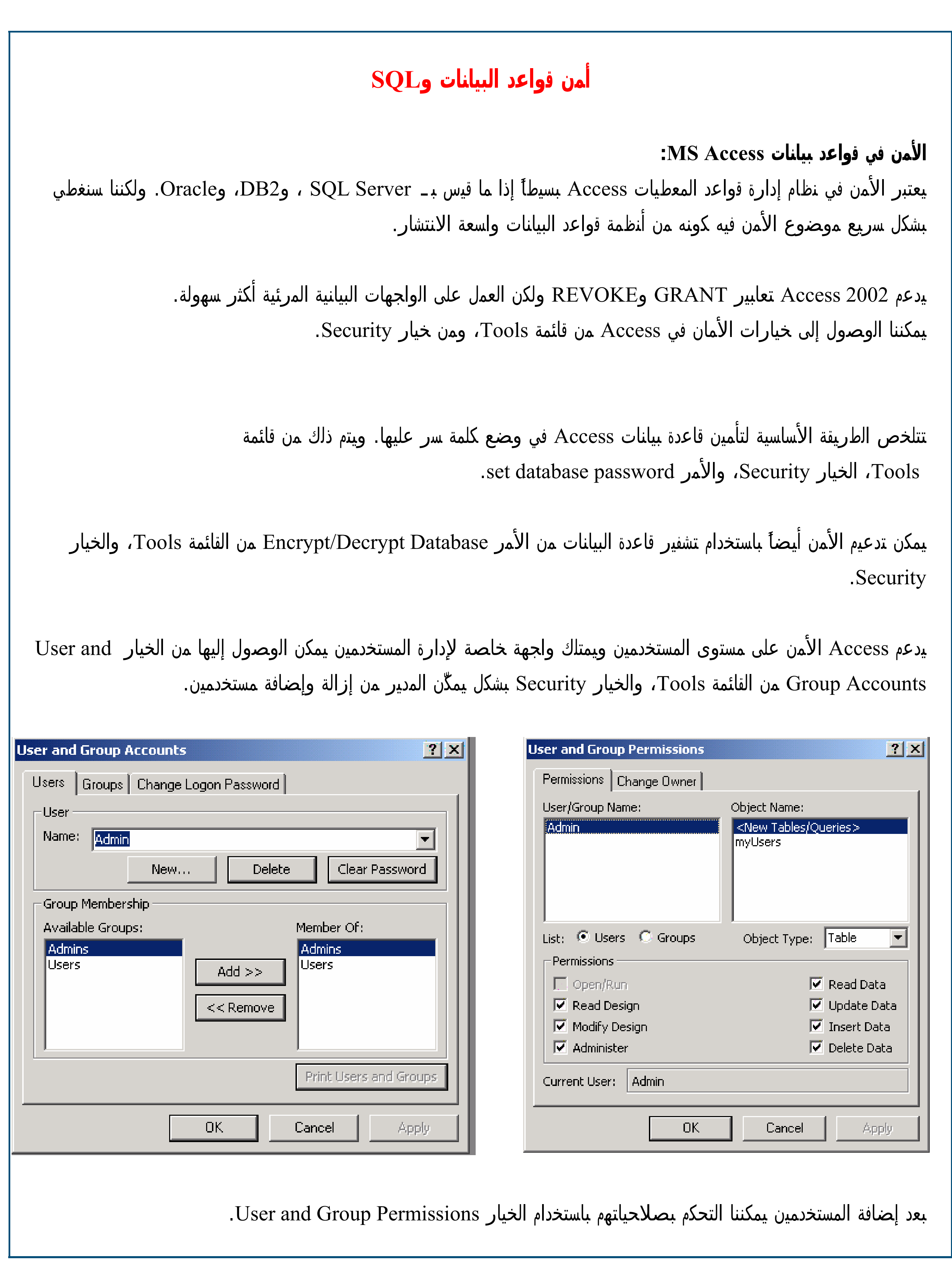
يعتبر الأمن في نظام إدارة قواعد المعطيات Access بسيطاً إذا ما قيس بـ SQL Server ، وDB2، وOracle. ولكننا سنغطي بشكل سريع موضوع الأمن فيه كونه من أنظمة قواعد البيانات واسعة الانتشار.

يدعم Access 2002 تعابير GRANT وREVOKE ولكن العمل على الواجهات البيانية المرئية أكثر سهولة. يمكننا الوصول إلى خيارات الأمان في Access من قائمة Tools، ومن خيار Security.

تتلخص الطريقة الأساسية لتأمين قاعدة بيانات Access في وضع كلمة سر عليها. ويتم ذلك من قائمة Tools، الخيار Security، والأمر set database password.

يمكن تدعيم الأمن أيضاً باستخدام تشفير قاعدة البيانات من الأمر Encrypt/Decrypt Database من القائمة Tools، والخيار Security.

يدعم Access الأمن على مستوى المستخدمين ويمتلك واجهة خاصة لإدارة المستخدمين يمكن الوصول إليها من الخيار User and Group Accounts من القائمة Tools، والخيار Security بشكل يمكّن المدير من إزالة وإضافة مستخدمين.



بعد إضافة المستخدمين يمكننا التحكم بصلاحياتهم باستخدام الخيار User and Group Permissions.

الأمن في قواعد بيانات MS Access:

يعتبر الأمن في نظام إدارة قواعد المعطيات Access بسيطاً إذا ما قيس بـ SQL Server ، وDB2، وOracle. ولكننا سنغطي بشكل سريع موضوع الأمن فيه كونه من أنظمة قواعد البيانات واسعة الانتشار.

يدعم Access 2002 تعابير GRANT وREVOKE ولكن العمل على الواجهات البيانية المرئية أكثر سهولة.
يمكننا الوصول إلى خيارات الأمان في Access من قائمة Tools، ومن خيار Security.

تتلخص الطريقة الأساسية لتأمين قاعدة بيانات Access في وضع كلمة سر عليها. ويتم ذلك من قائمة Tools، الخيار Security، والأمر set database password.

يمكن تدعيم الأمن أيضاً باستخدام تشفير قاعدة البيانات من الأمر Encrypt/Decrypt Database من القائمة Tools، والخيار Security.

يدعم Access الأمن على مستوى المستخدمين ويمتلك واجهة خاصة لإدارة المستخدمين يمكن الوصول إليها من الخيار User and Group Accounts من القائمة Tools، والخيار Security بشكل يمكّن المدير من إزالة وإضافة مستخدمين.

بعد إضافة المستخدمين يمكننا التحكم بصلاحياتهم باستخدام الخيار User and Group Permissions.

أمن قواعد البيانات وSQL

الأمن في قواعد بيانات SQL Server:

غالباً ما يتم في SQL Server، استخدام إجرائيات مُخزنة ومسبقة التعريف لإدارة المستخدمين ولتعريف عمليات إضافة مستخدم جديد أو تغيير كلمة السر أوتغيير صلاحيات مستخدم معين.

لإنشاء مستخدم جديد نستخدم الإجرائية المخزنة sp_addlogin. تحتاج هذه الإجرائية إلى اسم مستخدم وبصورة اختيارية كلمة سر ويتم استخدامها بالصيغة:

```
EXEC sp_addlogin ‘userName’, ‘userpassword’
```

لتغيير كلمة سر مستخدم نستعمل الإجرائية المخزنة sp_password. تأخذ هذه الإجرائية كمعامِلات: قيمة كلمة السر القديمة والكلمة الجديدة واسم المستخدم. تظهر الصيغة التالية كيفية استعمال الإجرائية sp_password:

```
EXEC sp_password 'oldpass', 'newpass', 'userName'
```

لإزالة مستخدم نستعمل ببساطة الإجرائية المخزنة sp_droplogin. تأخذ هذه الإجرائية كمعامل دخل اسم المستخدم المراد حذفه وهي بالصيغة:

```
EXEC sp_droplogin 'userName'
```

استخدام تحقق الهوية المتوفر في Windows في SQL Server:

يدعم SQL Server طريقة بديلة للتحقق من الهوية معروفة باسم الأمن المتكامل، حيث يتم ربط الوصول إلى قاعدة البيانات باستخدام التحقق من الهوية الخاص بـ Windows.

في حال اخترنا استخدام تحقق الهوية الخاص بـ Windows يمكننا أن نتعامل مع أسماء الحسابات بصورة مباشرة في عمليات توزيع الصلاحيات غير أننا لن نستطيع إنشاء مستخدم جديد أو أن نحذف مستخدم موجود.

للدلالة على اسم مستخدم خاص بنطاق نستعمل الصيغة domain\userName أما بالنسبة للحسابات المحلية فيتم الدلالة عليها بـ mymachine\userName.

يدعم SQL Server أيضاً ما يسمى بتحقق الهوية المختلط حيث يمكن استخدام تحقق الهوية الخاص بـ Windows مع تحقق الهوية الخاص بـ SQL Server.

مثال:

إذا أردنا إنشاء الحساب لاسم المستخدم 'Adel' بكلمة سر 'thepass' يمكننا كتابة الصيغة:

```
EXEC sp_addlogin 'Adel', 'thepass'
```

لإزالة هذا الحساب نستخدم الصيغة:

```
EXEC sp_droplogin 'Adel'
```

أمن قواعد البيانات وSQL

الأمن في قواعد بيانات SQL Server:

غالباً ما يتم في SQL Server، استخدام إجرائيات مُخزنة ومسبقة التعريف لإدارة المستخدمين ولتعريف عمليات إضافة مستخدم جديد أو تغيير كلمة السر أوتغيير صلاحيات مستخدم معين.

لإنشاء مستخدم جديد نستخدم الإجرائية المخزنة sp_addlogin

لتغيير كلمة سر مستخدم نستعمل الإجرائية المخزنة sp_password. تأخذ هذه الإجرائية كمعامِلات: قيمة كلمة السر القديمة والكلمة الجديدة واسم المستخدم.

لإزالة مستخدم نستعمل ببساطة الإجرائية المخزنة sp_droplogin. تأخذ هذه الإجرائية كمعامل دخل اسم المستخدم المراد حذفه

استخدام تحقق الهوية المتوفر في Windows في SQL Server:

يدعم SQL Server طريقة بديلة للتحقق من الهوية معروفة باسم الأمن المتكامل، حيث يتم ربط الوصول إلى قاعدة البيانات باستخدام التحقق من الهوية الخاص بـ Windows أي بالحسابات المحلية أوبحسابات النطاقات الخاصة بـ Windows.

في حال اخترنا استخدام تحقق الهوية الخاص بـ Windows يمكننا أن نتعامل مع أسماء الحسابات بصورة مباشرة في عمليات توزيع الصلاحيات غير أننا لن نستطيع إنشاء مستخدم جديد أو أن نحذف مستخدم موجود.

للدلالة على اسم مستخدم خاص بنطاق نستعمل الصيغة domain\userName أما بالنسبة للحسابات المحلية فيتم الدلالة عليها بـ mymachine\userName.

يدعم SQL Server أيضاً ما يسمى بتحقق الهوية المختلط حيث يمكن استخدام تحقق الهوية الخاص بـ Windows مع تحقق الهوية الخاص بـ SQL Server.

أمن قواعد البيانات وSQL

بعد إضافة الحسابات نستطيع البدء بتغيير الصلاحيات إذ لا يملك المستخدم الجديد أي صلاحيات.
يمكننا البدء بإعطاء المستخدم صلاحية الوصول إلى قواعد بيانات معينة باستخدام الإجرائية المخزنة sp_grantdbaccess. تسمح هذه الإجرائية لمستخدم ما بالاتصال بقاعدة البيانات ولكنها لا تمنحه أي صلاحية إضافية. تأخذ الإجرائية المخزنة الصيغة:

```
EXEC sp_grantdbaccess ‘userName’
```

مثال:

إذا أردنا إنشاء قاعدة البيانات myDatabase وإنشاء جدول myTable ضمنها ومنح المستخدم ‘Adel’ صلاحية الاتصال بقاعدة البيانات تلك نكتب الصيغة:

```
CREATE DATABASE myDatabase
GO
useMyDatabase
GO
CREATE TABLE myTable (myColumn varchar(10));
GO
EXEC sp_grantdbaccess ‘Adel’
```

لابد من ملاحظة أن الإجرائية المخزنة sp_grantdbaccess تعمل فقط مع قاعدة البيانات الحالية لذلك استخدمنا التعبير useMyDatabase لإعلام النظام أنه يتوجب عليه استخدام قاعدة البيانات myDatabase.

أما لحجب صلاحية الوصول إلى قاعدة البيانات لمستخدم ما نستخدم الإجرائية المخزنة sp_revokedbaccess وذلك وفق الصيغة:

```
EXEC sp_revokedbaccess ‘userName’
```

سنتناول في المرحلة التالية إعطاء المستخدم صلاحيات التحكم بأغراض قاعدة البيانات.

أمن قواعد البيانات وSQL

بعد إضافة الحسابات نستطيع البدء بتغيير الصلاحيات إذ لا يملك المستخدم الجديد أي صلاحيات.

يمكننا البدء بإعطاء المستخدم صلاحية الوصول إلى قواعد بيانات معينة باستخدام الإجرائية المخزنة sp_grantdbaccess. تسمح هذه الإجرائية لمستخدم ما بالاتصال بقاعدة البيانات ولكنها لا تمنحه أي صلاحية إضافية.

أما لحجب صلاحية الوصول إلى قاعدة البيانات لمستخدم ما نستخدم الإجرائية المخزنة sp_revokedbaccess.

سنتناول في المرحلة التالية إعطاء المستخدم صلاحيات التحكم بأغراض قاعدة البيانات.

إعداد صلاحيات مستوى الأغراض:

لنتمكن من إعداد الصلاحيات على SQL Server بصورة أمثلية يمكننا استخدام التعبيرات التي يوفرها المعيار SQL99 (GRANT, REVOKE) وذلك وفق الصيغة التي رأيناها مسبقاً.

مثال:

إذا أردنا إنشاء جدول myTest في قاعدة البيانات myDB وتخويل المستخدم 'Sami' إمكانية الاستعلام باستخدام SELECT أو التعديل باستخدام UPDATE، نكتب الصيغة:

```
CREATE DATABASE myDB
GO
USE myDB
GO
CREATE TABLE myTest (call1 INT);
GO
EXEC sp_grantdbaccess 'Sami'
GO
GRANT SELECT , UPDATE ON myTest TO 'Sami'
```

إذا أردنا منح المستخدم 'Sami' الحق بتمرير الصلاحيات التي حصل عليها بتعليمة GRANT، يمكننا استخدام WITH GRANT OPTION وذلك بالصيغة:

```
GRANT SELECT, UPDATE ON myTest TO 'Sami'  WITH GRANT OPTION
```

يمتلك SQL Server تعبير خاص معاكس لـ GRANT يدعى DENY الذي يقوم بتعديل وإعادة كتابة الصلاحيات من جديد.

أما تعبير REVOKE في SQL Server فيسبب إلغاء تأثير التعبير GRANT أو DENY. في هذه الحالة، يُعاد المستخدم إلى الوضعية التلقائية التي بدأ بها في حال استخدامنا REVOKE ALL.

فمثلاً لإزالة صلاحية استخدام الاستعلام بـ SELECT مع الجدول myTest نستخدم الصيغة:

```
REVOKE SELECT ON myTest FROM 'Sami'
```

أما إذا كنا قد استخدمنا الخيار WITH GRANT OPTION سنحصل على تعبير يقوم بإلغاء إمكانية الاستعلام من جميع الحسابات التي مُررت لها سماحية القراءة باستخدام التعبير CASCADE فتكون الصيغة:

```
REVOKE SELECT ON myTest FROM 'Sami'
```

إعداد سماحيات مستوى الأغراض:

لنتمكن من إعداد الصلاحيات على SQL Server بصورة أمثلية يمكننا استخدام التعبيرات التي يوفرها المعيار SQL99 (GRANT, REVOKE) وذلك وفق الصيغة التي رأيناها مسبقاً.

إذا أردنا منح المستخدم ‘Sami’ الحق بتمرير الصلاحيات التي حصل عليها بتعليمة GRANT، يمكننا استخدام WITH GRANT OPTION.

يمتلك SQL Server تعبير خاص معاكس لـ GRANT يدعى DENY الذي يقوم بتعديل وإعادة كتابة الصلاحيات من جديد.

أما تعبير REVOKE في SQL Server فيسبب إلغاء تأثير التعبير GRANT أو DENY. في هذه الحالة، يُعاد المستخدم إلى الوضعية التلقائية التي بدأ بها في حال استخدمنا REVOKE ALL.

أما إذا كنا قد استخدمنا الخيار WITH GRANT OPTION سنحصل على تعبير يقوم بإلغاء إمكانية الاستعلام من جميع الحسابات التي مُررت لها سماحية القراءة باستخدام التعبير CASCADE

أمن قواعد البيانات وSQL

استخدام الأدوار في MySQL:

عند تثبيت SQL Server نلاحظ أنه يتم تلقائياً إضافة مجموعة من الأدوار مسبقة الإعداد.

لنتمكن من إنشاء دور جديد نستخدم الإجرائية المخزنة sp_addrole ، تأخذ هذه الإجرائية اسم الدور الجديد كمعامل دخل.

بعد إضافة الدور سنحتاج إلى تحديد صلاحيات هذا الدور، لذا نستخدم التعبير GRANT مجدداً. فمثلاً لمنح صلاحية الاستعلام بـ SELECT عن الجدول myTable للدور myRole نكتب الصيغة:

```
GRANT SELECT ON myTable TO myRole;
```

لتحديد المستخدمين أصحاب الدور السابق نستخدم الصيغة:

```
EXEC sp_addrolemember myRole, userName;
```

لإزالة عضو من دور ما نستخدم الإجرائية المخزنة sp_droprolemember وذلك حسب الصيغة:

```
EXEC sp_droprolemember myRole, userName;
```

انتبه:

تعتبر تعبيرات DENY وREVOKE أقوى من التعبير GRANT أي أنك إذا حجبت باستخدام DENY سماحية ما عن مستخدم فهو لن يتمكن من أداء العمل المرتبط بالسماحية حتى لو كان عضواً في دور يسمح بأداء هذا العمل.

أمن قواعد البيانات وSQL

استخدام الأدوار في MySQL:

عند تثبيت SQL Server نلاحظ أنه يتم تلقائياً إضافة مجموعة من الأدوار مسبقة الإعداد.

لنتمكن من إنشاء دور جديد نستخدم الإجرائية المخزنة sp_addrole ، تأخذ هذه الإجرائية اسم الدور الجديد كمعامل دخل.

بعد إضافة الدور سنحتاج إلى تحديد صلاحيات هذا الدور، لذا نستخدم التعبير GRANT مجدداً.

أمن قواعد البيانات وSQL

إعداد سماحيات مستوى التصريح:

في SQL Server، يمكن للتعابير GRANT, DENY وREVOKE أن تتعامل مع صلاحيات مستوى التصريح حيث تساعد هذه السماحيات، المستخدم في التصريح عن أو في إنشاء أغراض جديدة في قاعدة البيانات.

تدعم SQL Server السماحيات التالية كسماحيات مستوى التصريح:

CREATE DATABASE, CREATE DEFAULT, CREATE FUNCTION, CREATE PROCEDURE, CREATE RULE, CREATE TABLE, CREATE VIEW, BACKUP DATABSE, BACKUP LOG.

مثال:

إذا أردنا إنشاء مستخدم test وإعطائه صلاحية إنشاء قاعدة بيانات جديدة نكتب الصيغة:

```
EXEC sp_addlogin 'test', 'testpass'
GRANT CREATE DATABASE TO 'test'
```

يمكننا هنا أيضاً استخدام الخيار WITH GRANT OPTION لتمكين تمرير الصلاحيات لمستخدم آخر أيضاً.

إعداد سماحيات مستوى التصريح:

في SQL Server، يمكن للتعابير GRANT, DENY وREVOKE أن تتعامل مع صلاحيات مستوى التصريح حيث تساعد هذه السماحيات، المستخدم في التصريح عن أو في إنشاء أغراض جديدة في قاعدة البيانات.

تدعم SQL Server السماحيات التالية كسماحيات مستوى التصريح:

CREATE DATABASE, CREATE DEFAULT, CREATE FUNCTION, CREATE PROCEDURE, CREATE RULE, CREATE TABLE, CREATE VIEW, BACKUP DATBSE, AND BACKUP LOG.

أمن قواعد البيانات وOracle

المحطة التالية هي ORACLE حيث سنرى ما هي التعابير والآليات التي تبنتها oracle لإدارة المستخدمين، وللتحقق من الهوية ولإدارة السماحيات.

إدارة المستخدمين والتحقق من الهوية:

توفر Oracle بعد التنصيب، بصورة تلقائية، حسابين هما SYS وSYSTEM.

لنتمكن من إدارة المستخدمين في oracle علينا الدخول بحساب SYSTEM لأنه الحساب الذي يملك الصلاحيات الخاصة بتلك العمليات.

- لإنشاء مستخدم جديد في Oracle نستعمل التعبير CREATE USER الذي يتطلب كمعاملات اسم المستخدم وكلمة المرور الخاصة به ، وإذا لم يتم تحديد كلمة السر سيقوم Oracle بجعلها تلقائياً simplepassword.

تستعمل CREATE USER الصيغة:

```
CREATE USER user_name IDENTIFIED BY user_password;
```

- لتعديل كلمة السر الخاصة بحساب ما نستخدم التعبير ALTER USER حيث يستعمل هذا التعبير الصيغة:

```
ALTER USER user_name IDENTIFIED BY new_password;
```

يمكننا عن طريق هذا التعبير أيضاً القيام بإقفال حساب ما وإيقاف صلاحياته دون حذفه أو تفعيل حساب تم إقفاله وذلك بالصيغة:

```
ALTER USER user_name ACCOUNT LOCK;
```

```
ALTER USER user_name ACCOUNT UNLOCK;
```

- أما لإزالة حساب مستخدم ما يكفي استخدام التعبير DROP USER الذي يأخذ الصيغة:

```
DROP USER user_name
```

مثال:

إذا أردنا إنشاء المستخدم test في قاعدة بيانات Oracle بكلمة سر تلقائية ثم تغييرها إلى sss ثم إزالة المستخدم نكتب الصيغة:

```
CREATE USER test;
ALTER USER test IDENTIFIED BY sss;
DROP USER test;
```

أمن قواعد البيانات و Oracle

المحطة التالية هي ORACLE حيث سنرى ما هي التعابير والآليات التي تبنتها oracle لإدارة المستخدمين، وللتحقق من الهوية ولإدارة السماحيات.

إدارة المستخدمين والتحقق من الهوية:

توفر Oracle بعد التنصيب، بصورة تلقائية، حسابين هما SYS وSYSTEM.

لنتمكن من إدارة المستخدمين في oracle علينا الدخول بحساب SYSTEM لأنه الحساب الذي يملك الصلاحيات الخاصة بتلك العمليات.

- لإنشاء مستخدم جديد في Oracle نستعمل التعبير CREATE USER الذي يتطلب كمعاملات اسم المستخدم وكلمة المرور الخاصة به ، وإذا لم يتم تحديد كلمة السر سيقوم Oracle بجعلها تلقائياً simplepassword.
- لتعديل كلمة السر الخاصة بحساب ما نستخدم التعبير ALTER USER
- يمكننا عن طريق هذا التعبير أيضاً القيام بإقفال حساب ما وإيقاف صلاحياته دون حذفه أو تفعيل حساب تم إقفاله
- أما لإزالة حساب مستخدم ما يكفي استخدام التعبير DROP USER

قبل الدخول في موضوع إدارة الصلاحيات والأدوار في ORACLE، من المفيد التنويه إلى أنّ التعبير ALTER USER يمنح الكثير من الخيارات مثل الإنهاء الفوري لفعالية كلمة السر وذلك بالصيغة:

```
ALTER USER user_name PASSWORD EXPIRE;
```

و ابتداءً من النسخة Oracle8 تمت إضافة ما يسمى PROFILE الذي يصف مجموعة الخصائص المراد تطبيقها على مستخدم ما. يمكن إجراء عملية الربط تلك أيضاً بواسطة التعبير ALTER USER وذلك بالصيغة:

```
ALTER USER user_name PROFILE my_profile;
```

لإنشاء Profile نستخدم الصيغة:

```
CREATE PROFILE my_profile LIMIT what_to_limit
```

يمكن من أن تأخذ what_to_limit أحد أو مجموعة من الخيارات التالية:

FAILED_LOGIN_ATTEMPTS تحدد عدد المحاولات الفاشلة قبل إقفال الحساب
PASSWORD_LIFE_TIME تحدد عمر كلمة السر دون تغييرها.
PASSWORD_REUSE_TIME تحدد الأيام الواجب انقضاؤها قبل إعادة استخدام كلمة سر سابقة
PASSWORD_REUSE_MAX تحدد عدد مرات تغير كلمة السر قبل إمكانية إعادة استخدام كلمة سر سابقة
PASSWORD_LOCK_TIME تحدد عدد الأيام التي سيبقى فيها الحساب مغلقاً بعد محاولات دخول فاشلة
PASSWORD_GRACE_TIME تحدد فترة التسامح بعد انقضاء فترة استعمال كلمة سر ما
PASSWORD_VERIFY_FUNCTION تحدد النص البر مجي الذي سيتحقق من التعقيد المطلوب لكلمة السر

لحذف PROFILE نستخدم ببساطة الصيغة:

```
DROP PROFILE my_profile;
```

مثال:

إذا أردنا كتابة الصيغة التي تقوم بإنشاء Profile يحدد عمر كلمة السر بـ 30 يوماً وتخصيص المستخدم Sara به نكتب الصيغة:

```
CREATE PROFILE my_profile LIMIT
PASSWORD_LIFE_TIME 30;
ALTER USER sara PROFILE my_profile;
```

القسم الثالث عشر

إدارة الأمن في قواعد البيانات باستخدام SQL (تتمة)

الكلمات المفتاحية:

الأمن ، التحقق من الهوية ، الصلاحيات، حساب ، مستخدم، الدور، مجموعة، صيغة، تعبير، مُختَطة، مستوى التصريح، مستوى الإغراض.

ملخص:

سنغطي في هذا القسم الأفكار الرئيسية المتعلقة بتعليمات SQL الخاصة بأمن قاعدة البيانات. كما سنستعرض طرائق تطبيقها في قواعد البيانات المختلفة.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- كيفية إدارة الأمن في DB2
- كيفية إدارة الأمن في MySQL
- كيفية إدارة الصلاحيات في ORACLE (تتمة)
- المُختَطات وإنشاؤها

أمن قواعد البيانات في ORACLE

أنهينا الجلسة السابقة بالحديث عن كيفية إدارة المستخدمين، وإدارة كلمة السر، وغيرها من المعلومات المتعلقة بحساب المستخدِم، في نظام إدارة قواعد المعطيات Oracle، سنتابع في هذه الجلسة شرح أسلوب إدارة Oracle للصلاحيات.

تحديد الصلاحيات في Oracle:

لتحديد الصلاحيات في Oracle نستخدم التعبيرين GRANT وREVOKE. لكن قبل البدء في استخدام هذين لتعبيرين، لابد لنا أن نتعرف على الأدوار المسبقة التعريف.

يوفر Oracle مجموعة من الأدوار مسبقة التعريف أهمها:

- CONNECT الذي يسمح للمستخدمين بالاتصال بقاعدة البيانات.
- RESOURCE الذي يسمح للمستخدمين بإنشاء جداولهم وإجرائياتهم وأغراض قاعدة البيانات الأخرى الخاصة بهم.
- DBA الذي تُقدم للمستخدمين صلاحيات كاملة.

إذا أردنا أن نقدم لمستخدم ما myUser صلاحيات الاتصال وإمكانية إنشاء أغراض والتحكم بها، يمكننا كتابة الصيغة:

```
GRANT CONNECT,RESOURCE TO myUser;
```

لحجب أحد الأدوار عن مستخدم ما، نستعمل التعبير REVOKE. فإذا أردنا مثلاً، حجب الدورRESOURCE عن المستخدم myUser يكفي كتابة الصيغة:

```
REVOKE RESOURCE FROM myUser;
```

لن يتمكن myUser الآن إلا من الاتصال بحسابه في قاعدة البيانات عن طريق اسم الدخول وكلمة السر.

انتبه:

- لا يكفي استخدام الدور RESOURCE لأن الصلاحيات التي يحويها الدور CONNECT غير محتواة في الدور RESOURCE
- إن تخصيص مستخدم بالدور RESOURCE يمنحه إمكانية الحصول على مساحة غير محددة لإنشاء جداوله وأغراض قاعدة البيانات الخاصة به، مما يفتح المجال لاستهلاك جميع الموارد دون أي اعتراض من قاعدة البيانات.

أمن قواعد البيانات في ORACLE

أنهينا الجلسة السابقة بالحديث عن كيفية إدارة المستخدمين في Oracle وطريقة ربط حساب ما بطابع معين لتحديد معلومات تتعلق بكلمة السر وفترة استخدامها، وغيرها من المعلومات المتعلقة بحساب المستخدِم.

سنتابع في هذه الجلسة شرح كيفية إدارة Oracle للصلاحيات.

تحديد الصلاحيات في Oracle:

لتحديد الصلاحيات في Oracle نستخدم التعبيرين GRANT وREVOKE.

قبل البدء باستخدام هذين لتعبيرين لابد لنا أن نتعرف على الأدوار مسبقة التعريف في Oracle.

يوفر Oracle مجموعة من الأدوار مسبقة التعريف أهمها:

- CONNECT الذي يسمح للمستخدمين بالاتصال بقاعدة البيانات.
- RESOURCE الذي يسمح للمستخدمين بإنشاء جداولهم وإجرائياتهم وأغراض قاعدة البيانات الأخرى الخاصة بهم.
- DBA الذي تقدم للمستخدمين صلاحيات كاملة.

أمن قواعد البيانات في ORACLE

إسناد أو حجب صلاحيات التعامل مع أغراض قواعد البيانات:

لإسناد صلاحية التعامل مع غرض من أغراض قاعدة البيانات، نعتمد على الصيغة:

```
GRANT privilege_type ON resource TO user_name;
```

يدعم Oracle الصلاحيات التالية:

- ALTER: التي تسمح بتعديل بنية جدول باستخدام تعبير ALTER TABLE.
- DELETE وINSERT وSELECT وUPDATE: التي يمكن تطبيقها على الجداول أو المناظير.
- INDEX: التي تسمح بإنشاء الفهارس في قاعدة البيانات.
- REFRENCES: التي تسمح بإدراج سجل يحتوي مفتاح أجنبي (أي ثانوي) من جدول آخر لا يستطيع المستخدم الاستعلام منه.
- EXECUTE: التي تسمح بتنفيذ الإجرائيات المُخزنة، أو التوابع، أوالوصول إلى أي برنامج محدد في طرد ما.

مثال:

سنقوم أولاً بالاتصال بقاعدة البيانات باستخدام التعبير CONNECT، اعتماداً على الصيغة:

```
CONNECT myUser/myPassword;
```

سنحاول الآن تنفيذ استعلام SELECT على أحد أغراض قاعدة البيانات وليكن الجدول test مثلاً. لكن علينا أن ننتبه أنه إذا أردنا الوصول إلى أحد أغراض قاعدة البيانات التي قام مستخدم آخر بإنشائها، وجب علينا وضع اسم المستخدم صاحب الغرض قبل اسم الغرض.

```
SELECT * FROM otherUser.test
```

سنحصل على إجابة بعدم وجود جدول بهذا الاسم، لأن myUser لا يملك صلاحية تنفيذ الاستعلام على الجدول test التابع للمستخدم otherUser. لذا لابد لنا من إعادة الاتصال بقاعدة البيانات باسم otherUser:

```
CONNECT otherUser/otherPass
```

ثم إعطاء سماحية الاستعلام باستخدام الصيغة:

```
GRANT SELECT ,INSERT ON test TO myUser;
```

الآن يمكننا إعادة الاتصال من جديد عن طريق حساب المستخدم myUser وتطبيق استعلامنا بنجاح.
يدعم Oracle الصلاحيات التالية:

- ALTER: التي تسمح بتعديل بنية جدول باستخدام تعبير ALTER TABLE.
- DELETE وINSERT وSELECT وUPDATE: التي يمكن تطبيقها على الجداول أو المناظير.
- INDEX: التي تسمح بإنشاء الفهارس في قاعدة البيانات.
- REFRENCES: التي تسمح بإدراج سجل يحتوي مفتاح أجنبي (أي ثانوي) من جدول آخر لا يستطيع المستخدم الاستعلام منه.
- EXECUTE: التي تسمح بتنفيذ الإجرائيات المُخزنة، أو التوابع، أوالوصول إلى أي برنامج محدد في طرد ما.

أمن قواعد البيانات في ORACLE

الأدوار غير مسبقة التعريف:

ذكرنا سابقاً أن Oracle يحتوي على مجموعة من الأدوار المُسبقة التعريف وذكرنا مهمة كل منها، ولكن هذا لا يعني أننا لا نستطيع إنشاء أدوار جديدة.

نستخدم التعبير CREATE ROLE لإنشاء دور جديد في Oracle وذلك اعتماداً على الصيغة:

```
CREATE ROLE role_name;
```

أما إذا أردنا حذف دور ما، فيمكننا استخدام التعبير DROP ROLE وذلك اعتماداً على الصيغة:

```
DROP ROLE role_name;
```

يجب الانتباه إلى أننا نحتاج إلى الدخول إلى قاعدة البيانات عبر حساب المستخدم SYSTEM لإضافة دور.

بعد عملية إضافة دور سنحتاج إلى تحديد الصلاحيات المحتواة في هذا الدور. لذا نستخدم التعبير GRANT لإضافة الصلاحيات على دور ما:

```
GRANT privilege_type ON resource TO role_name
```

الآن يكفي تخصيص المستخدم بدور ما ليرث جميع الصلاحيات الخاصة بهذا الدور وذلك اعتماداً على الصيغة:

```
GRANT my_role TO unser_name;
```

مثال:

إذا أردنا إنشاء دور جديد باسم customer وأردنا إضافة صلاحية قراءة الجدول Products لهذا الدور، ثم أردنا إنشاء المستخدم Rami بكلمة سر rpass وتخصيصه بالدور Customer فما علينا إلا كتابة الصيغة:

```
CREATE ROLE customer ;
GRANT SELECT ON Products TO customer;
CREATE USER Ram IDENTFIED BY rpass;
GRANT customer TO Rami;
```

الأدوار غير مسبقة التعريف:

ذكرنا سابقاً أن Oracle يحتوي على مجموعة من الأدوار المُسبقة التعريف وذكرنا مهمة كل منها، ولكن هذا لا يعني أننا لا نستطيع إنشاء أدوار جديدة.

نستخدم التعبير CREATE ROLE لإنشاء دور جديد في Oracle. أما إذا أردنا حذف دور ما، فيمكننا استخدام التعبير DROP ROLE.

يجب الانتباه إلى أننا نحتاج إلى الدخول إلى قاعدة البيانات عبر حساب المستخدم SYSTEM لإضافة دور.

بعد عملية إضافة دور سنحتاج إلى تحديد الصلاحيات المحتواة في هذا الدور. لذا نستخدم التعبير GRANT لإضافة الصلاحيات على دور ما

من ثم يكفي تخصيص المستخدم بدور ما ليرث جميع الصلاحيات الخاصة بهذا الدور.

أمن قواعد البيانات في ORACLE

تحديد صلاحيات على مستوى التصريح في Oracle:

يمكن التعامل أيضاً مع الصلاحيات على مستوى التصريح في Oracle باستخدام GRANT وREVOKE.

إذ يمكنك استخدام الصيغة:

```
GRANT privilege_type TO user_name;
```

أو

```
REVOKE privilege_type FROM user_name;
```

تتعامل GRANT وREVOKE مع قائمة صلاحيات طويلة تخص مستوى التصريح نذكر أهمها:

CREATE DATABASE وCREATE TABLE وALTER TABLE وDROP TABLE

عموماً، يمكن أن يُستخدم التعبير GRANT مع الخيار WITH GRANT OPTION مما يساعد المستخدم على تمرير صلاحياته إلى مستخدم آخر.

تكون معظم هذه الصلاحيات مشمولة بالدور المسبق التعريف RESOURCE. فإذا رغبنا باعتماد الحل الأسهل يمكننا استخدام الدور RESOURCE، أما إذا أردنا أن نكون صارمين من ناحية تحديد الصلاحيات بدقة، فمن الأفضل تحديدها بالتفصيل.

مثال:

إذا أردنا إنشاء المستخدم test مع إعطائه الصلاحيات اللازمة لإنشاء قواعد البيانات والجداول، وحجب صلاحية حذف الجداول التي يمكن أن يكون قد حصل عليها في عبارة سابقة. نكتب الصيغة:

```
CREATE USER test;
GRANT CREATE TABLE,CREAT DATABASE TO test;
REFOKE DROP TABLE FROM test;
```

تحديد صلاحيات على مستوى التصريح في Oracle:

يمكن التعامل أيضاً مع الصلاحيات على مستوى التصريح في Oracle باستخدام GRANT وREVOKE.

تتعامل GRANT وREVOKE مع قائمة صلاحيات طويلة تخص مستوى التصريح نذكر أهمها:

CREATE DATABASE وCREATE TABLE وALTER TABLE وDROP TABLE

عموماً، يمكن أن يُستخدم التعبير GRANT مع الخيار WITH GRANT OPTION مما يساعد المستخدم على تمرير صلاحياته إلى مستخدم آخر.

تكون معظم هذه الصلاحيات مشمولة بالدور المسبق التعريف RESOURCE. فإذا رغبنا باعتماد الحل الأسهل يمكننا استخدام الدور RESOURCE، أما إذا أردنا أن نكون صارمين من ناحية تحديد الصلاحيات بدقة، فمن الأفضل تحديدها بالتفصيل.

أمن قواعد البيانات في DB2

إدارة المستخدمين في قواعد بيانات DB2:

بعكس قواعد البيانات الأخرى، لا تملك DB2 آلية منفصلة لإضافة المستخدمين وإدارتهم، بل تعتمد وبصورة كاملة على نظام التشغيل. فحين نريد إضافة مستخدم إلى قاعدة البيانات، علينا إنشاء مستخدم بالآلية التي يوفرها نظام التشغيل. عندئذ نستطيع إسناد أو حجب أية صلاحية للمستخدم الذي تم إنشاؤه.

تحديد السماحيات على مستوى الأغراض في قاعدة البيانات DB2:

لا تختلف طريقة DB2 في إسناد وحجب الصلاحيات عن قواعد البيانات الأخرى، فهي تعتمد أيضاً على GRANT وREVOKE وتستخدم نفس الصيغ:

```
GRANT privilege_type ON resource TO user_name;
```

و

```
REVOKE privilege_type ON resource FROM user_name
```

تتضمن الصلاحيات الممكنة: ALTER, DELETE, SELECT, UPDATE, INSERT, INDEX, REFRENCE, EXECUTE والتي قمنا بتوضيحها مسبقاً.

كما تُعطى صلاحية CONTROL لحساب مدير قاعدة البيانات ومدير النظام فقط وتُعبر عن صلاحية إسناد صلاحيات (عدا CONTRTOL) لمستخدمين آخرين.

مثال:

إذا أردنا منح الحساب Maria الصلاحيات الخاصة بالإدراج والحذف من الجدول test بالإضافة إلى حجب إمكانية تعديل بنية الجدول،

يتوجب علينا أولاً السماح للحساب بالاتصال بقاعدة البيانات:

```
GRANT CONNECT ON DATABASE TO USER Maria
```

ثم يتوجب علينا ثانياً إسناد صلاحيات الإدراج والحذف وذلك بالصيغة:

```
GRANT DELETE,INSERT ON test TO Maria;
```

ثم يتوجب علينا ثالثاً حجب التعديل على بنية الجدول وذلك الصيغة:

```
REVOKE ALTER ON test FROM Maria;
```

إدارة المستخدمين في قواعد بيانات DB2:

بعكس قواعد البيانات الأخرى، لا تملك DB2 آلية منفصلة لإضافة المستخدمين وإدارتهم، بل تعتمد وبصورة كاملة على نظام التشغيل. فحين نريد إضافة مستخدم إلى قاعدة البيانات، علينا إنشاء مستخدم بالآلية التي يوفرها نظام التشغيل. عندئذ نستطيع إسناد أو حجب أية صلاحية للمستخدم الذي تم إنشاؤه.

تحديد سماحيات مستوى الأغراض في قاعدة البيانات DB2:

لا تختلف طريقة DB2 في إسناد وحجب الصلاحيات عن قواعد البيانات الأخرى فهي تعتمد أيضاً على GRANT وREVOKE

تتضمن الصلاحيات الممكنة: ALTER, DELETE, SELECT, UPDATE, INSERT, INDEX, REFRENCE, EXECUTE والتي قمنا بتوضيحها مسبقاً.

كما تُعطى صلاحية CONTROL لحساب مدير قاعدة البيانات ومدير النظام فقط وتُعبر عن صلاحية إسناد صلاحيات (عدا CONTRTOL) لمستخدمين آخرين.

أمن قواعد البيانات في DB2

استخدام المجموعات في DB2:

لا تدعم DB2 البنية التقليدية للمستخدمين والأدوار، لذلك تعتمد على نظام التشغيل في إنشاء مجموعات المستخدمين.

بعد إنشاء مجموعة مستخدمين عن طريق نظام التشغيل، يمكننا تحديد الصلاحيات الخاصة بها من خلال الصيغة:

```
GRANT privilege_name ON resource TO GROUP myGroup;
```

مثال:

إذا أردنا منح المجموعة Special صلاحية إدراج وحذف بيانات من الجدول Customer، علينا أولاً أن نمنح هذه المجموعة صلاحية الاتصال بقاعدة البيانات وذلك بالصيغة:

```
GRANT CONNECT ON DATABASE TO GROUP Special;
GRANT INSERT , DELETE ON Customer TO Special;
```

في حال كانت صلاحيات مستخدم أعلى من صلاحيات المجموعة التي ينتمي إليها، يحتفظ المستخدم بإمكانية أداء العمليات المخصصة له.

لحجب صلاحية معينة نستخدم التعبير REVOKE. ففي مثالنا السابق، نستخدم الصيغة التالية إذا أردنا حجب عملية الاستعلام بـ SELECT عن المجموعة:

```
REVOKE SELECT ON Customer FROM Special;
```

في حال أردنا حجب جميع الصلاحيات عن مستخدم يمكننا استخدام الصيغة:

```
REVOKE ALL PRIVILEGES ON resource from userName;
```

أمن قواعد البيانات في DB2

استخدام المجموعات في DB2:

لا تدعم DB2 البنية التقليدية للمستخدمين والأدوار، لذلك تعتمد على نظام التشغيل في إنشاء مجموعات المستخدمين.

بعد إنشاء مجموعة مستخدمين عن طريق نظام التشغيل، يمكننا تحديد الصلاحيات الخاصة بها.

في حال كانت صلاحيات مستخدم أعلى من صلاحيات المجموعة التي ينتمي إليها، يحتفظ المستخدم بإمكانية أداء العمليات المخصصة له.
لحجب صلاحية معينة نستخدم التعبير REVOKE.

أمن قواعد البيانات في DB2

تحديد الصلاحيات على مستوى التصريح:

كغيرها من قواعد البيانات تستطيع DB2 أن تحجب الصلاحيات على مستوى التصريح مثل الصلاحيات الخاصة بإنشاء الجداول وقواعد البيانات... وغيرها.

تستعمل DB2 الصيغة:

```
GRANT privilege_type ON DATABESE TO user_name;
```

مثال:

إذا أردنا منح المستخدم Farid صلاحية إنشاء جدول في قاعدة البيانات نكتب الصيغة:

```
GRANT CREATE TAB ON DATABASE TO Farid;
```

أما إذا أردنا الحصول على صلاحيات عالية وإدارة كاملة لقاعدة البيانات يمكننا استخدام السماحية DBADM وتكون الصيغة من الشكل:

```
GRANT DBADM ON DATABASE TO Farid;
```

أمن قواعد البيانات في DB2

تحديد الصلاحيات على مستوى التصريح:

كغيرها من قواعد البيانات تستطيع DB2 أن تحجب الصلاحيات على مستوى التصريح مثل الصلاحيات الخاصة بإنشاء الجداول وقواعد البيانات... وغيرها.

أمن قواعد البيانات في MySQL

توفر قواعد بيانات MySQL مجالاً ضيقاً من خيارات الأمن ولاتدعم الأدوار.

تخزن سماحيات المستخدمين في جداول خاصة بالأمن تسمى GRANT TABLES يتم إنشاؤها تلقائياً عند تثبيت قواعد بيانات MySQL.

هناك ستة جداول من نمط GRANT TABLES: user وtables_priv وcolumn_priv وdb وfunc وhost يخدَم كل جدول غرض مختلف ونستطيع التعديل عليها يدوياً لحجب أو إسناد صلاحيات معينة.

يمكن إضافة مستخدم جديد بإضافة سجل إلى الجدول user وذلك بالصيغة:

```
USE mysql
Insert Into user values(‘localhost’, ‘userName’, PASSWORD
(‘simplepass’) ,’Y’, ’Y’, ’Y’, ’Y’, ’Y’, ’Y’, ’Y’, ’Y’, ’Y’, ’Y’,
’Y’, ’Y’, ’Y’, ’Y’,);
FLUSH PRIVILEGES;
```

- تضيف الصيغة السابقة المستخدم userName@localhost إلى قاعدة البيانات.
- تعبر الحقول التي حددناها بالقيمة ‘Y’ عن الصلاحيات المتاحة.
- استخدمنا التابع PASSWORD لتحويل كلمة السر إلى الصيغة المشفرة.

لابد لنا من استخدام الأمر FLUSH PRIVILEGES بعد أي تعديل على الجداول الخاصة بالصلاحيات، لإعادة تحميلها. إذ تتم قرائة هذه الجداول لمرة واحدة عند إقلاع MySQL ولا تتم إعادة تحميلها.

نستطيع أن نحقق الغاية السابقة باستخدام التعبير GRANT الذي يربط صلاحيات بمستخدم ما.
في حال استعمال التعبيرGRANT مع مستخدم غير موجود، يتم إنشاؤه تلقائياً.

أمن قواعد البيانات في MySQL

توفر قواعد بيانات MySQL مجالاً ضيقاً من خيارات الأمن ولاتدعم الأدوار.

تخزن سماحيات المستخدمين في جداول خاصة بالأمن تسمى GRANT TABLES يتم إنشاؤها تلقائياً عند تثبيت قواعد بيانات MySQL.

هناك ستة جداول من نمط GRANT TABLES: user وtables_priv وcolumn_priv وdb وfunc وhost يخدَم كل جدول غرض مختلف ونستطيع التعديل عليها يدوياً لحجب أو إسناد صلاحيات معينة.

لابد لنا من استخدام الأمر FLUSH PRIVILEGES بعد أي تعديل على الجداول الخاصة بالصلاحيات، لإعادة تحميلها. إذ تتم قرائة هذه الجداول لمرة واحدة عند إقلاع MySQL ولا تتم إعادة تحميلها.

نستطيع أن نحقق الغاية السابقة باستخدام التعبير GRANT الذي يربط صلاحيات بمستخدم ما.

في حال استعمال التعبيرGRANT مع مستخدم غير موجود، يتم إنشاؤه تلقائياً.

أمن قواعد البيانات في MySQL

للتعبير GRANT الصيغة:

```
GRANT privilege_type ON recourse TO userName [IDENTIFIED BY
password];
```

عند استعمال GRANT لإدارة الصلاحيات يجب علينا أيضاً الانتباه إلى استخدام الخيار FLUSH PRIVILEGES.

مثال:

إذا أردنا إعطاء المستخدم Tony صلاحية استعراض سجلات الجدول myTable، يمكننا استخدام الصيغة:

```
GRANT SELECT ON myTable TO Tony;
```

في حال رغبنا بإعطاء جميع الصلاحيات على جميع قواعد البيانات والجداول التي تنتمي إليها نستخدم الصيغة:

```
GRANT ALL ON *.* TO Tony@locahost;
```

لحجب صلاحية حذف جدول ما من قاعدة البيانات وليكن Test نكتب الصيغة:

```
REVOKE DROP ON dbName.Test FROM Tony@localhost;
```

لحجب صلاحية الاستعلام بـ SELECT عن جميع الجداول في جميع قواعد البيانات نستخدم الصيغة:

```
REVOKE SELECT ON *.* FROM Tony@localhost;
```

أما إذا أردنا معرفة الصلاحيات التي يتمتع بها مستخدم ما، نستعمل التعبير SHOW GRANTS وذلك بالصيغة:

```
SHOW GRANTS FOR Tony@localhost;
```

مُختَطات قواعد البيانات

تحدثنا عن الأغراض في قاعدة البيانات وذكرنا أنها تشمل أغراض قواعد البيانات، والجداول، والجداول المؤقتة، والفهارس، والقيود وأغفلنا مُختَطات قواعد البيانات بسبب حاجتنا إلى استخدام الصلاحيات في بناء هذه المُختَطات.

تُعرف مُختَطات قاعدة البيانات بأنها عبارة عن مجموعة من التعابير التي تشمل بناء الجداول والمناظير وتحديد السماحيات عليها.

تدعم كل من قواعد بيانات SQL Server, Oracle وDB2، المُختَطات.

تحتوي المُختَطات التعابير CREATE TABLE، CREATE VIEW، GRANT وتعمل ككتلة واحدة على شكل مناقلة.

لا يتم فصل التعابير في المُختَطات بفواصل منقوطة لأنها تُعامل ككتلة واحدة.

لإنشاء المُختَطات نستخدم التعبير CREATE SCHEMA AUTHORIZATION وذلك بالصيغة:

```
CREATE SCHEMA AUTHORIZATION schemaName
-- create tables
-- create view
-- grant permissions;
```

مثال:

إذا أردنا إنشاء المُختَطة mySchema التي تُنشئ الجدول Customers والمنظار Names وتخول المستخدم Sami الإستعلام بـ SELECT عن المنظار Names نكتب الصيغة:

```
CREATE SCHEM AUTHORIZATION mySchema
CREATE TABLE Customers
            (ID INT PRIMARY KEY NOT NULL , Name varchar(50))
CREATE VIEW Names AS SELECT Name FROM Customers
GRANT SELECT ON Names TO Sami;
```

لايؤثر تسلسل التعابير في جسم المُختَطة. إذ سيتم تنفيذ جميع التعابير حتى لو ظهرت بتسلسل غير مطابق لتسلسلها المنطقي.

مخططات قواعد البيانات

تحدثنا عن الأغراض في قاعدة البيانات وذكرنا أنها تشمل أغراض قواعد البيانات، والجداول، والجداول المؤقتة، والفهارس، والقيود وأغفلنا مُختَطات قواعد البيانات بسبب حاجتنا إلى استخدام الصلاحيات في بناء هذه المُختَطات.

تُعرف مُختَطات قاعدة البيانات بأنها عبارة عن مجموعة من التعابير التي تشمل بناء الجداول والمناظير وتحديد السماحيات عليها.

تدعم كل من قواعد بيانات SQL Server, Oracle وDB2، المُختَطَات.

تحتوي المُختَطَات التعابير CREATE TABLE، CREATE VIEW، GRANT وتعمل ككتلة واحدة على شكل مناقلة.

لا يتم فصل التعابير في المُختَطَات بفواصل منقوطة لأنها تُعامل ككتلة واحدة.

لإنشاء المُختَطَات نستخدم التعبير CREATE SCHEMA AUTHORIZATION

لايؤثر تسلسل التعابير في جسم المُختَطَة. إذ سيتم تنفيذ جميع التعابير حتى لو ظهرت بتسلسل غير مطابق لتسلسلها المنطقي.

القسم الرابع عشر

القادحات في قواعد البيانات

الكلمات المفتاحية:

قادح ، حدث، مستوى التعبير، مستوى السجل، مستوى قاعدة البيانات، مستوى المخطط، تفعيل، تعطيل، تعديل، إنشاء، صيغة.

ملخص:

تُعتبر القادحات من الآليات التي تزود المبرمج بتقنياتٍ تساعده على التحكم بشكلٍ أفضل بالأحداث وبالعمليات في قواعد البيانات.
ستغطي هذه الجلسة المعلومات المتعلقة باستخدام القادحات في SQL Server وOracle.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- مفهوم القادحات بشكل عام
- القادحات وإدارتها في قواعد بيانات SQL Server
- القادحات وإدارتها في قواعد بيانات Oracle



القادحات

تُعتبر القادحات وسيلةً لتأمين تطبيق قواعد العمل وتكامل البيانات واتساقها وتماسكها داخل قاعدة البيانات. فهي نوع خاص من الإجرائيات المُخزنة المرتبطة بجدول معين، يتم تنفيذها بصورة آلية عند أي تعديل على هذا الجدول.

يمكننا باستخدام معالجات الأحداث أن نؤتمت بدء تنفيذ نص SQL برمجي بجعله يتجاوب مع أحداث معينة.

من الضروري عند التعامل مع القادحات تحديد الجدول المراد ربط القادح معه وتحديد الحدث الذي سيتم إطلاقه. فهل نريد مثلاً للقادح أن يعمل حين نضيف سجلاً ما إلى جدول، أم عند حذف سجلٍ ما من الجدول، أم عند تعديله، أم عند ظهور تركيبة معينة من هذه الأحداث.
كما نحتاج إلى اتخاذ قرار بشأن لحظة الإطلاق، وفيما إذا كانت قبل أو بعد حدوث. فهل سيتم مثلاً تحفيز القادح قبل عملية إضافة السجل أم بعدها.

بإمكاننا تنفيذ قادحات تعاكس الأحداث التي سببت إطلاقها، كأن ننفذ قادح ما قبل تنفيذ حذف سجل بحيث نلغي عملية تنفيذ الحذف كلياً.

تشبه القادحات الإجرائيات المخزنة من حيث النص البرمجي الذي بإمكانها احتوائه ومن حيث قدرتها على تمرير معاملات دخل وخرج.

نستطيع إنشاء أكثر من قادح على جدول واحد في قاعدة البيانات ويمكن لقادح أن يطلق قادح آخر أو أن يطلق نفسه بصورة عودية.

كالعادة، هناك العديد من الإختلافات في تنفيذ القادحات بين أنظمة إدارة قواعد البيانات بالرغم من أن المبدأ يبقى واحداً، وهو ما سنستعرضه في القسم الحالي والقسم الذي يليه لتغطية موضوع القادحات في قواعد البيانات SQL server, DB2,Oracle,mySQL.

القادحات

أنواع القادحات:

تصنف القادحات إلى نوعين أساسيين وذلك تبعاً لعدد مرات تنفيذ نص القادح عندما يؤثر الحدث على مجموعة من السجلات:

- **القادحات على مستوى التعبير:** حيث يتم إطلاق هذا النوع من القادحات مرة واحدة لكل تعبير INSERT أو DELETE أو UPDATE حتى ولو قام هذا الأخير بالتأثير على أكثر من سجل.

  فإذا كان لدينا قادح يرتبط بتعبير يسبب حذف مئة سجل، سيتم تنفيذ النص البرمجي للقادح مرة واحدة.

- **القادحات على مستوى السجل:** في هذا النوع من القادحات يتم تنفيذ النص البرمجي للقادح مع كل سجل تؤثر فيه إحدى عبارات UPDATE أو DELETE أو INSERT. فإذا كان لدينا قادح من هذا النوع مرتبط بتعبير UPDATE يقوم بتعديل مئة سجل، سيتم تنفيذ النص البرمجي للقادح مئة مرة.

يمكننا جعل أي من نوعي القادحات ينفذ قبل أو بعد أو عوضاً عن الحدث الذي قام بإطلاقها.

يجب أن نأخذ بالحسبان أن القادحات التي تُنفذ قبل الحدث، تنفذ دون تحقق على الحدث نفسه، بينما القادحات التي تُنفذ بعد الحدث، لن تُنفذ ما لم يتم تنفيذ التعبير الممثل للحدث بصورة صحيحة. لذلك غالباً ما تستخدم هذه القادحات في تقييم نجاح التعابير والتعليمات.

تدعم بعض قواعد البيانات القادحات التي تتجاوب مع أحداث مختلفة، كإنشاء جدول أو تحديثه أو تسجيل دخول مستخدم ولكننا لن نخوض في هذه التفاصيل مع ملاحظة أن الفكرة العامة لا تتغير بالنسبة لتلك الأحداث.

إنشاء قادح:

يتم إنشاء قادح باستخدام التعبير CREATE TRIGGER.

يحدد المعيار SQL-99 الصيغة الخاصة بإنشاء قادح كما يلي، ولكن هذه الصيغة غير مطبقة بشكلها المعياري في أي نوع من أنواع أنظمة إدارة قواعد البيانات، لذا سنكتفي بسردها للمهتمين فقط:

```
CREATE TRIGGER trigger_name
{ BEFORE | AFTER }
{[DELETE] | [INSERT] | [UPDATE]
{OF column [,…n]} ON table_name
 [ROW] [AS] new_name [REFERNCING {OLD [ROW][AS] old_name | NEW
OLD TABLE [AS] old_name | NEW TABLE [AS] new_name}]
[FOR EACH { ROW | STATEMENT }\
[WHEN (condition)]
--sql code block
```

القادحات

أنواع القادحات:

تصنف القادحات إلى نوعين أساسيين وذلك تبعاً لعدد مرات تنفيذ نص القادح عندما يؤثر الحدث على مجموعة من السجلات:

- **القادحات على مستوى التعبير:** حيث يتم إطلاق هذا النوع من القادحات مرة واحدة لكل تعبير INSERT أو DELETE أو UPDATE حتى ولو قام هذا الأخير بالتأثير على أكثر من سجل.
  فإذا كان لدينا قادح يرتبط بتعبير يسبب حذف مئة سجل، سيتم تنفيذ النص البرمجي للقادح مرة واحدة.

- **القادحات على مستوى السجل:** في هذا النوع من القادحات يتم تنفيذ النص البرمجي للقادح مع كل سجل تؤثر فيه إحدى عبارات UPDATE أو DELETE أو INSERT. فإذا كان لدينا قادح من هذا النوع مرتبط بتعبير UPDATE يقوم بتعديل مئة سجل، سيتم تنفيذ النص البرمجي للقادح مئة مرة.

يمكننا جعل أي من نوعي القادحات ينفذ قبل أو بعد أو عوضاً عن الحدث الذي قام بإطلاقها.

يجب أن نأخذ بالحسبان أن القادحات التي تُنفذ قبل الحدث، تنفذ دون تحقق على الحدث نفسه، بينما القادحات التي تُنفذ بعد الحدث، لن تُنفذ ما لم يتم تنفيذ التعبير الممثل للحدث بصورة صحيحة. لذلك غالباً ما تستخدم هذه القادحات في تقييم نجاح التعابير والتعليمات.

تدعم بعض قواعد البيانات القادحات التي تتجاوب مع أحداث مختلفة، كإنشاء جدول أو تحديثه أو تسجيل دخول مستخدم ولكننا لن نخوض في هذه التفاصيل مع ملاحظة أن الفكرة العامة لا تتغير بالنسبة لتلك الأحداث.

إنشاء قادح:

يتم إنشاء قادح باستخدام التعبير CREATE TRIGGER.
يحدد المعيار SQL-99 الصيغة الخاصة بإنشاء قادح، ولكن هذه الصيغة غير مطبقة بشكلها المعياري في أي نوع من أنواع أنظمة إدارة قواعد البيانات.

القادحات

استخدام القادحات:

تُستخدم القادحات في تطبيقات مختلفة ولكن أهم استخداماتها هي:

1- تأمين التكاملية المرجعية والتي عادةً ما تتم باستخدام المفاتيح الأجنبية (التي ندعوها أيضاً المفاتيح الثانوية). ففي بعض الحالات، نستخدم القادحات إذا لم تكن هذه المفاتيح قوية بالدرجة الكافية لتنفيذ هذا العمل، كحالة التكاملية المرجعية بين الجداول في أكثر من قاعدة بيانات، أوعلى أكثر من مخدّم قاعدة بيانات.
2- تطبيق قواعد العمل المعقدة لضمان عدم وجود أية عملية تقوم بكسر قواعد العمل أو تكاملية البيانات.
3- متابعة وتسجيل كل العمليات التي تتم على الجداول في قاعدة البيانات.
4- محاكاة عمل القيد CHECK بين الجداول وقواعد البيانات ومخدمات قواعد البيانات.
5- اعتراض أوامر المستخدم واستبدالها، حيث يمكننا مقاطعة الأوامر أو الأفعال التي يقوم بها المستخدم واستبدالها بأفعال أخرى.

استخدام القادحات في قواعد بيانات SQL Server:

تدعم قواعد بيانات SQL Server القادحات على مستوى التعبير فقط ولا تدعم القادحات على مستوى السجل. فإذا استخدمنا تعبير يقوم بحذف عشرة سجلات، سوف يتم تنفيذ القادح مرة واحدة لكامل العبارة.

لا يدعم SQL Server القادحات التي تسبق الحدث أي التي تستخدم التعبير BEFORE، ويدعم فقط القادحات اللاحقة. وابتداءاً من النسخة SQL Server 2000 بات يدعم القادحات البديلة أي التي تستخدم التعبير INSTEAD.

استخدام القادحات في قواعد بيانات SQL Server

إنشاء قادح:

لإنشاء قادح نستخدم التعبير CREATE TRIGGER وذلك بالصيغة:

```
CREATE TRIGGER triggerName ON tableName FOR [|INSTEAD] action AS
-- procedureBody;
```

فمثلاً، إذا أردنا إنشاء قادح مرتبط بحدث إضافة سجل إلى الجدول Test الذي يحتوي حقل القيمة Value وحقل الرقم التسلسلي ID، بحيث يقوم هذا القادح بتسجيل عملية إضافة السجل إلى الجدول Audit، نكتب الصيغة:

```
CREATE TRIGGER myTrigger ON Test FOR Insert
AS
DECALRE @newValue varchar(50)
SELECT @newValue = value FROM Inserted
Insert Into Audit (newValue) Values (@newValue);
```

نلاحظ في الصيغة السابقة أننا قمنا باستخدام القادح بعد إضافة سجل إلى الجدول Test، كما نلاحظ أننا استفدنا من قيم الجدول Inserted وهو جدول تم إنشاؤه تلقائياً يحتوي السجل الذي يتم العمل عليه وله بنية مطابقة لبنية الجدول الذي تم ربط القادح به.

عند ربط قادح بعملية حذف سجل، يتم أيضاً إنشاء جدول باسم Deleted يحتوي السجل الذي تم حذفه له نفس بنية الجدول الذي تم ربط القادح به.

تكمن مشكلة هذا الحل في أننا لن نستطيع تسجيل عملية إدراج مجموعة من السجلات باستخدام تعبير واحد. فإذا تم استخدام التعبير INSERT INTO…SELECT، لن يتم تنفيذ القادح سوى مرة واحدة. إذ سبق وذكرنا أن SQL Server لاتدعم القادحات مستوى السجل.

مثال:

إذا أردنا إنشاء قادح ليقوم بتسجيل عمليات الحذف من جدول Test تكون الصيغة:

```
CREATE TRIGGER logDelete
ON Test FOR DELETE AS
DECALRE @newValue varchar(50)
SELECT @deletedValue = value FROM Deleted
Insert Into Audit (newValue) Values (@deletedValue);
```

تعديل قادح:

لتعديل قادح نستخدم التعبير ALTER TRIGGER وذلك بالصيغة:

```
ALTER TRIGGER triggerName ON tableName FOR [|INSTEAD] action AS
-- procedureBody;
```

مثال:

إذا أردنا دمج عمليتي الحذف والإدراج في قادح واحد يمكن تعديل القادح السابق myTrigger بحيث نحصل على الصيغة:

```
ALTER TRIGGER myTrigger ON Test
FOR INSERT, DELETE
AS
IF EXISTS (SELECT 1 FROM Inserted)
BEGIN
INSERT INTO Audit (newValue) SELECT Inserted.Value FROM Inserted
END
ELSE IF EXISTS (SELECT 1 FROM Deleted)
BEGIN
INSERT INTO Audit (newValue) SELECT Deleted.Value FROM Deleted
END;
```

ستعمل الصيغة السابقة حتى في حالة إدراج أكثر من سجل في الجدول Test، ولكن عملها لن يكون بفضل القادحات من مستوى السجل (لأنها غير مدعومة أصلاً)، بل بفضل التعبير الذي تم اختياره في عملية إدراج القيم ضمن الجدول Audit، حيث يقوم التعبير بإدراج كامل السجلات المدخلة والتي تم نسخها إلى الجدول Deleted أو Inserted إلى الجدول Audit.

حذف قادح:

لحذف قادح نستخدم التعبير DROP TRIGGER وذلك بالصيغة:

```
DROP TRIGGER triggerName;
```

مثال:

لحذف القادح باسم myTrigger الذي أنشأناه مسبقاً نستخدم الصيغة:

```
DROP TRIGGER myTrigger;
```

تعديل قادح:

لتعديل قادح نستخدم التعبير ALTER TRIGGER.

حذف قادح:

لحذف قادح نستخدم التعبير DROP TRIGGER.

استخدام القادحات في قواعد بيانات SQL Server

حالة UPDATE:

لم نذكر حالة ربط القادح في SQL Server، بتعديل سجل، ولم نذكر شيئاً عن وجود ما يسمى Updated على غرار Deleted وInserted وذلك لعدم وجوده أصلاً.

فعلياً، عند ربط قادح بحدث UPDATE، يتم ملء كلاً من الجدولين Inserted وDeleted.

فإذا أردنا تعديل المثال السابق ليشمل حالات الإدراج والحذف والتعديل يمكننا كتابة الصيغة:

```
ALTER TRIGGER myTrigger ON Test FOR INSERT, DELETE, UPDATE AS
IF EXISTS (SELECT 1 FROM Inserted) AND EXISTS (SELECT 1 FROM Deleted)
BEGIN
INSERT INTO Audit (newValue) SELECT D.Value FROM Deleted D JOIN
Inserted I on D.Value = I.Value
END
ELSE IF EXISTS (SELECT 1 FROM Inserted)
BEGIN
INSERT INTO Audit (newValue) SELECT Inserted.Value FROM Inserted
END
ELSE IF EXISTS (SELECT 1 FROM Deleted)
BEGIN
INSERT INTO Audit (newValue) SELECT Deleted.Value FROM Deleted
END;
```

نلاحظ هنا أننا اختبرنا وجود سجلات في كلا الجدولين Inserted وDeleted لنتحقق بأن العملية هي عملية تعديل، ثم قمنا بربط السجلات في الجدولين بـ JOIN لاستخلاص القيم التي نريد إدراجها.

تفعيل وتعطيل القادحات:

قد يلزمنا في بعض الأحيان ولأسباب خاصة بتحسين الأداء، تعطيل عمل بعض القادحات بشكل مؤقت، وبالأخص تلك التي تؤدي مهام المتابعة وتسجيل الحركة.

يتم تفعيل القادحات تلقائياً بعد إنشائها، ولكنك تستطيع تعطيلها باستخدام التعبير ALTER TABLE وذلك اعتماداً على الصيغة:

```
ALTER TABLE table_name DISABLE TRIGGER trigger_name;
```

ولإعادة تفعيل قادح تم تعطيله يمكننا استخدام الصيغة:

```
ALTER TABLE table_name ENABLE TRIGGER trigger_name;
```

أما إذا أردنا تعطيل أو تفعيل جميع القادحات الخاصة بجدول ما، فنستخدم التعبير ALL بدلاً من اسم القادح. تصبح الصيغة عندها:

```
ALTER TABLE table_name [ENABLE | DISABLE] TRIGGER ALL;
```

استخدام القادحات في قواعد بيانات SQL Server

حالة UPDATE:

لم نذكر حالة ربط القادح في SQL Server، بتعديل سجل، ولم نذكر شيئاً عن وجود ما يسمى Updated على غرار Deleted وInserted وذلك لعدم وجوده أصلاً.

فعلياً، عند ربط قادح بحدث UPDATE، يتم ملء كلاً من الجدولين Inserted وDeleted.

تفعيل وتعطيل القادحات:

قد يلزمنا في بعض الأحيان ولأسباب خاصة بتحسين الأداء، تعطيل عمل بعض القادحات بشكل مؤقت، وبالأخص تلك التي تؤدي مهام المتابعة وتسجيل الحركة.

يتم تفعيل القادحات تلقائياً بعد إنشائها، ولكنك تستطيع تعطيلها باستخدام التعبير ALTER TABLE

أما إذا أردنا تعطيل أو تفعيل جميع القادحات لجدول ما نستخدم التعبير ALL بدلاً من اسم القادح.

استخدام القادحات في قواعد بيانات Oracle

يدعم نظام إدارة قواعد البيانات Oracle القادحات السابقة واللاحقة والبديلة لحدث ما، حيث تُنفذ القادحات اللاحقة بعد تنفيذ الحدث، أما القادحات السابقة والبديلة فيتم تنفيذها قبل تنفيذ الحدث.

كما يدعم نظام إدارة قواعد البيانات Oracle القادحات على مستوى التعبير والقادحات على مستوى السجل.

غالباً ما تُستخدم القادحات على مستوى التعبير لتنفيذ اختبارات على البيانات قبل إجراء العمليات عليها، وهي لا تستخدم فعلياً للتحكم بالبيانات التي تم إدراجها أو تعديلها أو حذفها، إذ غالباً ما تُترك هذه المهمة للقادحات على مستوى السجل.

يتم تفعيل القادحات على مستوى السجل عند كل تأثير على سجل، ويتكرر تفعيل القادح بعدد السجلات المتأثرة بالحدث (إدراج، حذف، تعديل).

يُشار إلى القيم التي تم التأثير عليها بالحدث DELETE بالسابقة:**OLD** والقيم التي تم إدراجها بالحدث INSERT بالسابقة:**NEW** بشكل مشابه لما رأيناه بالنسبة لـ Inserted وDeleted في SQL Server.

يدعم Oracle أيضاً ما يسمى بقادحات مستوى قاعدة البيانات، وقادحات مستوى المخطط والتي تكون مفيدة في حالة رغبنا بأتمتة أعمال الصيانة ومتابعة الأحداث والأخطاء.

تُستخدم قادحات مستوى المخطط مع أحداث من الشكل CREATE TABLE، وALTER TABLE وDROP TABLE، في حين تُستعمل قادحات مستوى قاعدة البيانات مع عمليات تسجيل الدخول والخروج وعمليات التشغيل وإيقاف التشغيل.

استخدام القادحات في قواعد بيانات Oracle

إنشاء قادح في قواعد بيانات Oracle:

لإنشاء قادح في قواعد بيانات Oracle نستخدم التعبير CREATE TRIGGER أيضاً وذلك بالصيغة:

```
CREATE TRIGGER trigger_name
[AFTER | BEFORE INSTEAD] [INSERT | DELETE | UPDATE]
ON table_name
-- trigger_body;
```

مثال:

لنقم بإنشاء الجدول myTable الذي يحتوي حقل ID وحقل Comment.

```
CREATE TABLE myTable
(ID INT PRIMARY KEY NOT NULL , Comment varchar(50))
```

إذا أردنا إنشاء قادح myTrigger لرصد عمليات إضافة السجلات على الجدول myTable وتسجيل رقم التعليق ونوع العملية في جدول Audit يمكننا كتابة الصيغة:

```
CREATE TRIGGER myTrigger
AFTER INSERT ON myTable
FOR EACH ROW
BEGIN
INSERT INTO Audit (ID , operationType) Values (:NEW.ID , 'INSERT')
END;
```

نلاحظ أننا استخدمنا التعبير FOR EACH ROW ليقوم بالمرور على جميع القيم التي تم إدراجها في الجدول myTable في حال تم إدراج عدة سجلات بتعبير واحد.

استخدمنا أيضاً:NEW.ID للدلالة على قيمة ID في السجل الجديد الذي تم إدراجه.

استخدام القادحات في قواعد بيانات Oracle

تعديل قادح في قواعد بيانات Oracle:

لتعديل قادح في قواعد بيانات Oracle نستخدم التعبير CREATE OR REPLACE TRIGGER وذلك بالصيغة:

```
CREATE OR REPLACE TRIGGER trigger_name
 INSTEAD] [INSERT | DELETE | UPDATE] |[AFTER | BEFORE
ON table_name
-- trigger_body;
```

مثال:

لنجرب الآن تعديل المثال السابق ليدعم الحذف والتعديل:

```
OR REPLACE TRIGGER myTrigger  CREATE
AFTER INSERT OR DELETE OR UPDATE ON myTable
FOR EACH ROW
BEGIN
IF INSERTING THEN
INSERT INTO Audit
(ID , operationType) Values (:NEW.ID , 'INSERT');
ELSIF DELETING THEN
INSERT INTO Audit
(ID , operationType) Values (:OLD.ID , 'DELETE');

ELSIF UPDATEING THEN
INSERT INTO Audit
(ID , operationType) Values (:OLD.ID , 'UPDATE');

END IF;

END;
```

نلاحظ في هذا المثال أننا ربطنا هذا القادح بـ INSERT وDELETE وUPDATE ورأينا أن Oracleيسمح لنا بتحديد نوع تعبير SQL الذي تسبب بتفعيل القادح وذلك بإعطاء القيمة True ل INSERTING أو DELETING أو UPDATING.

في حال تنفيذ عمليةUpdate يتم إعطاء قيمة ل.**:NEW** ول.:**OLD** كما ذكرنا القيم من.:OLD هي القيم قبل التعديل ومن.:NEW القيم بعد التعديل.

حذف قادح:

يمكننا حذف قادح باستخدام التعبير DROP TRIGGER كما هو الحال في SQLserver وذلك بالصيغة:

```
DROP TRIGGER trigger_name;
```

استخدام القادحات في قواعد بيانات Oracle

تعديل قادح في قواعد بيانات Oracle:

لتعديل قادح في قواعد بيانات Oracle نستخدم التعبير CREATE OR REPLACE TRIGGER

حذف قادح:

يمكننا حذف قادح باستخدام التعبير DROP TRIGGER كما هو الحال في SQLserver

استخدام القادحات في قواعد بيانات Oracle

تطبيق الشروط على قادح في قواعد بيانات Oracle:

يتيح نظام إدارة قواعد المعطيات Oracle تطبيق شروط تحدد فيما إذا كان سيتم تفعيل القادح أم لا، وذلك باستخدام التعبير WHEN مع التعبير CREATE TRIGGER.

مثال:

إذا أردنا إنشاء قادح بحيث يقوم بفتح حساب جديد في الجدول Accounts عند إيداع مبلغ ما في الجدول Depositions، إلا إذا كان اسم الشخص 'sami'، نكتب الصيغة:

```
CREATE OR REPLACE TRIGGER myTrigger
AFTER INSERT ON Depositions
FOR EACH ROW
WHEN (NEW.name!='sami')
BEGIN
INSERT INTO Accounts (AccountName) Values (:NEW.name);
END;
```

كيفية توليد الترقيم التلقائي عن طريق القادحات:

تساعد القادحات في تشغيل نص برمجي قبل أو بعد عملية إدراج، مما يساعد في محاكاة آلية الترقيم التلقائي الموجودة في SQL Server وDB2 وMySQL وAccess إذا كنا لا نريد إدخال هذه القيمة يدوياً عند كل إدراجنا لسجل جديد.

أول ما يجب فعله هو إنشاء متتالية بالصيغة:

```
CREATE SEQUENCE mySequence;
```

ومن ثم، يتوجب علينا إنشاء قادح يقوم بتوليد الرقم التسلسلي التالي قبل إدراج السجل. تتم العملية قبل إدراج السجل لأننا نحتاج إلى توليد الرقم التسلسلي قبل إضافة السجل.

لهذا الغرض نكتب الصيغة:

```
CREATE OR REPLACE TRIGGER myTrigger
BEFORE INSERT ON myTable
FOR EACH ROW
BEGIN
SELECT mySequence.NEXTVAL INTO:NEW.myTable FROM DUAL
END;
```

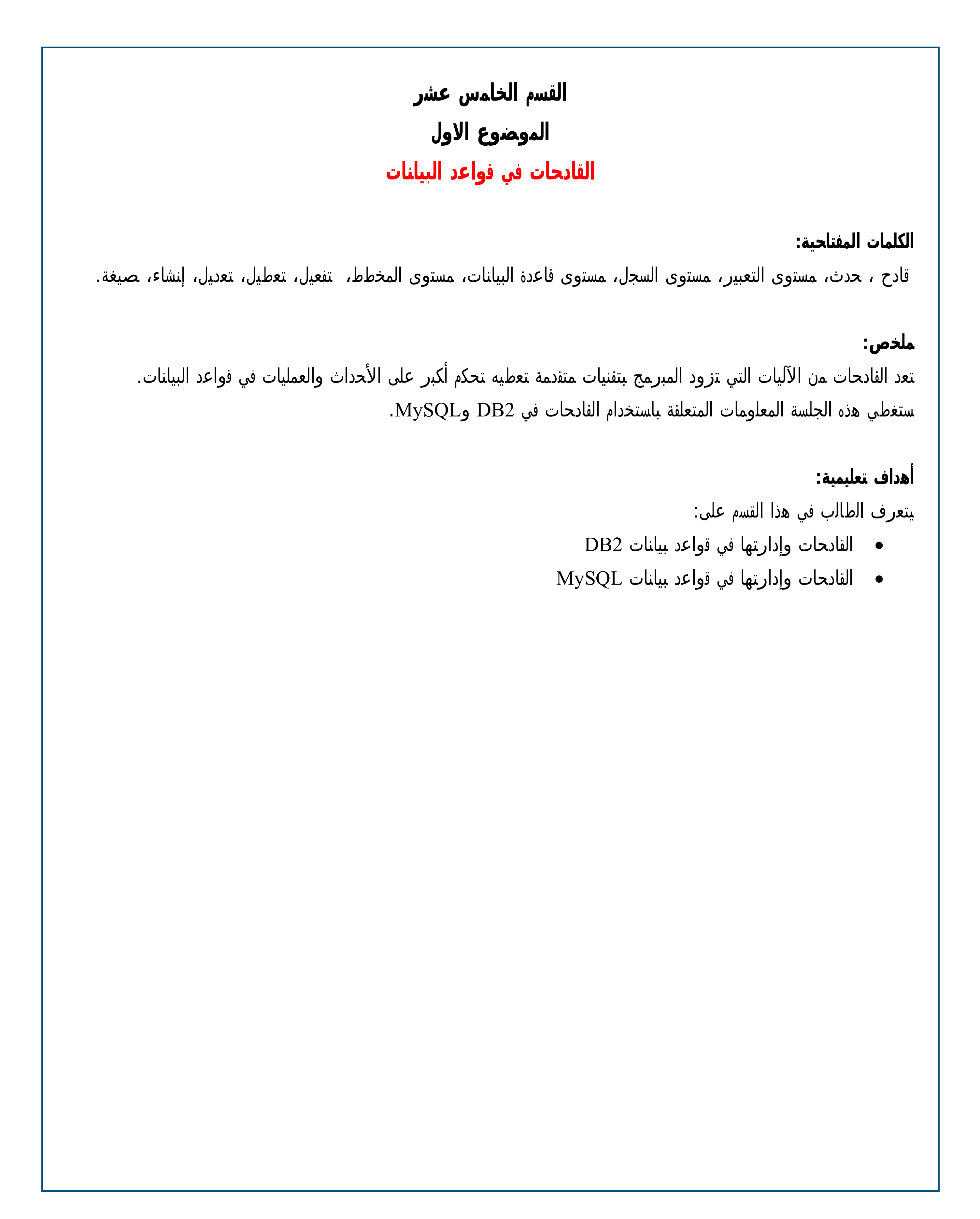
استخدام القادحات في قواعد بيانات Oracle

تطبيق الشروط على قادح في قواعد بيانات Oracle:

يتيح نظام إدارة قواعد المعطيات Oracle تطبيق شروط تحدد فيما إذا كان سيتم تفعيل القادح أم لا، وذلك باستخدام التعبير WHEN مع التعبير CREATE TRIGGER.

كيفية توليد الترقيم التلقائي عن طريق القادحات:

تساعد القادحات في تشغيل نص برمجي قبل أو بعد عملية إدراج، مما يساعد في محاكاة آلية الترقيم التلقائي الموجودة في SQL Server وDB2 وMySQL وAccess إذا كنا لا نريد إدخال هذه القيمة يدوياً عند كل إدراجنا لسجل جديد.



القسم الخامس عشر

الموضوع الاول

القادحات في قواعد البيانات

الكلمات المفتاحية:

قادح ، حدث، مستوى التعبير، مستوى السجل، مستوى قاعدة البيانات، مستوى المخطط، تفعيل، تعطيل، تعديل، إنشاء، صيغة.

ملخص:

تعد القادحات من الآليات التي تزود المبرمج بتقنيات متقدمة تعطيه تحكم أكبر على الأحداث والعمليات في قواعد البيانات.
ستغطي هذه الجلسة المعلومات المتعلقة باستخدام القادحات في DB2 وMySQL.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- القادحات وإدارتها في قواعد بيانات DB2
- القادحات وإدارتها في قواعد بيانات MySQL

القادحات في قواعد بيانات DB2

نتابع حديثنا عن القادحات في قواعد البيانات ونتناول في هذه الوحدة القادحات في قواعد بيانات DB2.

إنشاء قادح:

يشبه تعبير إنشاء قادح في DB2 للتعبير المستخدم في Oracle ولكن بفروق بسيطة. تأخذ الصيغة الشكل:

```
CREATE TRIGGER trigger_name [AFTER |NOCASCADBEFORE | INSTEADOF] ON
table_name REFERENCING [OLD AS | NEW AS] refname
FOR EACH ROW MODE DB2SQL
trigger_body;
```

نلاحظ أن DB2 تستخدم قادحات تستعمل AFTER ويتم تنفيذها بعد تنفيذ عبارة الحدث الذي سببها، وذلك بعد التحقق من القيود وتنفيذ التعليمة بنجاح. كما تدعم DB2 القادحات التي تستعمل NOCASCADBEFORE، وINSTEADOF التي يتم تنفيذها قبل تنفيذ الحدث دون التأكد من أي قيود.

تمنع القادحات مع NOCASCADBEFORE الحدث الذي حفز القادح من تنفيذ أي قادح أخر.

كحال Oracle، تدعم DB2 القادحات على مستوى التعبير والقادحات على مستوى السجل.

كما نلاحظ أننا قمنا باستخدام التعبير REFERENCING لإعطاء قيم مؤقتة لمتحولات ستحتوي معلومات عن السجل أو الجدول قبل وبعد تنفيذ الحدث.

يستخدم التعبير OLD AS old_name لتحديد السجلات القديمة والتعبير NEW AS new_name لتحديد السجلات الجديدة.

مثال:

إذا أردنا كتابة صيغة لإنشاء قادح يقوم بإضافة سجل في الجدول Table2 لدى إدراج سجل في الجدول Table1 نكتب:

```
CREATE TRIGGER myTrigger
AFTER INSERT ON Table1 REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
INSERT INTO Table2 (operation , ID) Values ('INSERT', N.ID)
END;
```

انتبه:
لا بد دائماً من وضع التعبير MODE DB2SQL بالرغم من كونه الخيار الوحيد المتوفر.

القادحات في قواعد بيانات DB2

نتابع حديثنا عن القادحات في قواعد البيانات ونتناول في هذه الوحدة القادحات في قواعد بيانات DB2.

إنشاء قادح:
يشبه تعبير إنشاء قادح في DB2 للتعبير المستخدم في Oracle ولكن بفروق بسيطة.

القادحات في قواعد بيانات DB2

تعديل قادح:
لتعديل قادح نقوم بحذفه وإعادة إنشائه. لتنفيذ العملية نستخدم التعبير DROP TRIGGER وذلك بالصيغة:

```
DROP TRIGGER trigger_name;
```

ثم نقوم بإنشائه من جديد.

حالة DELETE وUPDATE:
رأينا في المثال السابق كيف يتم إنشاء قادح مرتبط بعملية إدراج سجل، ورأينا كيفية تعاملنا مع التعبير REFERENCING للوصول إلى قيم السجل الجديد، سنغطي فيما يلي حالة DELETE وUPDATE.

لنحاول إنشاء القادح myTrigger الذي يضيف القيم القديمة والحديثة للسجل الذي يتم تعديله من جدول myTable إلى جدول باسم Audit.
يحتوي الجدول myTable على حقل وحيد يسمى Value والجدول Audit على حقلين أحدهما oldValue والثاني newValue.

لتنفيذ العملية نكتب الصيغة:

```
CREATE TRIGGER myTrigger
AFTER UPDATE ON myTable REFERENCING OLD AS O NEWAS N
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
INSERT INTO Audit (oldValue, newValue) Values (O.Value, N.Value)
END;
```

القادحات في قواعد بيانات DB2

تعديل قادح:

لتعديل قادح نقوم بحذفه وإعادة إنشائه. لتنفيذ العملية نستخدم التعبير DROP TRIGGER.

حالة DELETE وUPDATE:

رأينا في المثال السابق كيف يتم إنشاء قادح مرتبط بعملية إدراج سجل، ورأينا كيفية تعاملنا مع التعبير REFERENCING للوصول إلى قيم السجل الجديد، سنغطي فيما يلي حالة DELETE وUPDATE.

القادحات في قواعد بيانات MySQL

تطبيق الشروط على قادح:

كحال Oracle نستطيع في DB2 استخدام الشروط للتحكم بتنفيذ القادح.

مثال:

نقوم بإدراج عمليات البيع في الجدول Sales بحيث يتم إدخال اسم السلعة واسم الزبون والجهة التي ينتمي إليها الزبون. لكن نريد إنشاء قادح يقوم بالتحقق من كون اسم الجهة هو Company ليقوم بإضافة سجل إلى الجدول Discounts.

لنقم أولاً بحذف القادح الذي أنشأناه بالصيغة:

```
DROP TRIGGER myTrigger;
```

ننشئ القادح بالصيغة:

```
CREATE TRIGGER myTrigger
AFTER INSERT ON Sales
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
WHEN (N.association='company')
BEGIN ATOMIC
INSERT INTO Discounts (userName) Values (N.userName)
END;
```

القادحات في قواعد بيانات MySQL

لم تكن القادحات مدعومة في قواعد بيانات MySQL قبل النسخة 5.0.2. وهي ما تزال محدودة بعض الشيء إذا ما قيست بمثيلاتها في Oracle وDB2.

تدعم MySQL القادحات السابقة واللاحقة التي يتم تحديدها بالتعابير BEFORE وAFTER.

إنشاء قادح:

يتم إنشاء قادح بالتعبير CREATE TRIGGER الذي يأخذ الصيغة:

```
CREATE TRIGGER trigger_name trigger_time trigger_event
    ON tbl_name
FOR EACH ROW trigger_stmt
```

مثال:

لنفترض أننا نريد إنشاء قادح يقوم بعملية التأكد من أن حساب الشخص ينخفض بمقدار المبالغ التي يقوم بدفعها.

```
CREATE TRIGGER checkBalance
AFTER  INSERT ON payments
FOR EACH ROW
BEGIN
UPDATE accounts
set Balance=Balance-NEW.paymentAmount where AccountID=NEW.ID
END;
```

حذف قادح:

لحذف قادح نستخدم نفس الصيغة المستعملة في قواعد البيانات الأخرى:

```
DROP TRIGGER trigger_name;
```

القادحات في قواعد بيانات MySQL

لم تكن القادحات مدعومة في قواعد بيانات MySQL قبل النسخة 5.0.2. وهي ما تزال محدودة بعض الشيء إذا ما قيست بمثيلاتها في Oracle وDB2.

تدعم MySQL القادحات السابقة واللاحقة التي يتم تحديدها بالتعابير BEFORE وAFTER.

إنشاء قادح:

يتم إنشاء قادح بالتعبير CREATE TRIGGER.

حذف قادح:

لحذف قادح نستخدم نفس الصيغة المستعملة في قواعد البيانات الأخرى وهي DROP TRIGGER.

القادحات في قواعد بيانات MySQL

يساعد تعبيرا OLD وNEW في الوصول إلى أعمدة السجلات التي أثر بها الحدث الذي حفز القادح.

في حالة القادح المرتبط بعملية إدراج سجل، نستخدم التعبير NEW أما في في حالة القادح المرتبط بعملية الحذف، فنستخدم التعبير OLD. كما يمكننا استخدام كل من OLD وNEW في حالة القادح المرتبط بعملية التعديل، للوصول إلى القيم قبل وبعد التعديل.

تبقى القيم المستعادة من OLD، للقراءة فقط ولا يمكن تعديلها. أما القيم المُستعادة من NEW فيمكن تخصيصها في حال كان القادح الذي نتعامل معه يستخدم BEFORE.

حالة UPDATE:

إذا أردنا إنشاء قادح يقوم بالتقاط حدث التعديل على جدول myTable ليسجل القيم قبل وبعد التعديل في الجدول Log، نكتب الصيغة:

```
CREATE TRIGGER myTrigger
AFTER  UPDATE ON myTable
FOR EACH ROW
BEGIN
INSERT INTO Log (oldValue, newValue) Values (OLD.Column1,
NEW.Column1)
END;
```

لنجرب صيغة أخرى تقوم بتعديل القيمة المدرجة في جدول وإضافة 10 تلقائياً عليها.

```
CREATE TRIGGER addTen
BEFORE INSERT ON numberTable
FOR EACH ROW
BEGIN
NEW.Number = NEW.Number +10
END;
```

ما تزال القادحات في MySQL غير ناضجة كونها لم تدعَم سوى مؤخراً ولكنها أثبتت فعاليتها.

القادحات في قواعد بيانات MySQL

يساعد تعبيرا OLD وNEW في الوصول إلى أعمدة السجلات التي أثر بها الحدث الذي حفز القادح.

في حالة القادح المرتبط بعملية إدراج سجل، نستخدم التعبير NEWن أما في في حالة القادح المرتبط بعملية الحذف، فنستخدم التعبير OLD. كما يمكننا استخدام كل من OLD وNEW في حالة القادح المرتبط بعملية التعدبل، للوصول إلى القيم قبل وبعد التعديل.

تبقى القيم المستعادة من OLD، للقراءة فقط ولا يمكن تعديلها. أما القيم المُستعادة من NEW فيمكن تخصيصها في حال كان القادح الذي نتعامل معه يستخدم BEFORE.

القسم الخامس عشر

الموضوع الثاني

التوابع المُعرفة من قبل المستخدم

الكلمات المفتاحية:

تابع، معامِل، دخل، خرج، إنشاء، صيغة.

ملخص:

تعرفنا في مراحل سابقة على التوابع المسبقة التعريف وسنتعرف في هذه الجلسة على التوابع المُعرفة من قبل المستخدم في قواعد البيانات المختلفة.

أهداف تعليمية:

يتعرف الطالب في هذا القسم على:

- التوابع وإنشاؤها في قواعد بيانات SQL Server
- التوابع وإنشاؤها في قواعد بيانات DB2
- التوابع وإنشاؤها في قواعد بيانات MySQL
- التوابع وإنشاؤها في قواعد بيانات Oracle

إنشاء التوابع

استعرضنا في الجلسات الماضية التوابع وأنواعها وميزنا التوابع التجميعية والتوابع الدرجية بأنواعها المختلفة ولكن ماذا لو أردنا إنشاء توابعنا الخاصة؟

يُعرف المعيار ANSI SQL-99 التعبير CREATE FUNCTION الذي يساعد على إنشاء التوابع.

تسمى التوابع التي يتم إنشاؤها بالتعبير CREATE FUNCTION، بالتوابع المُعرفة من قبل المستخدم أو User Defined Functions.

تأخذ الصيغة البسيطة العامة في أغلب قواعد البيانات الشكل:

```
CREATE FUNCTION function_name [(parameter_list)]
RETURNS data_type
-- SQL Statements
```

للخوض في التفاصيل لا بد لنا من الاطلاع على كيفية تعامل كل قاعدة بيانات مع التوابع.

استدعاء تابع:

يتم استدعاء التوابع المُعرفة من قبل المستخدم بنفس الطريقة المستخدمة مع التوابع الأخرى أي يمكننا بعد إنشاء التابع استدعاؤه بصورة فورية بالشكل:

```
SELECT function_name (parameter_list) AS Test;
```

التوابع في Access:

لا تدعم Access التوابع المعرفة من قبل المستخدم.

التوابع في SQL Server:

يستخدم SQL Server لإنشاء تابع من قبل مستخدم الصيغة التالية:

```
CREATE FUNCTION function_name [(parameter_list)]
RETURNS data_type
AS
BEGIN
SQL Statements
RETURN expression
END;
```

يمكن للتوابع في SQL أن تحتوي أكثر من تعبير ولكن يجب ألا تسبب أي تغيير على البيانات في قاعدة البيانات.

إنشاء التوابع

استعرضنا في الجلسات الماضية التوابع وأنواعها وميزنا التوابع التجميعية والتوابع الدرجية بأنواعها المختلفة ولكن ماذا لو أردنا إنشاء توابعنا الخاصة؟

يُعرف المعيار ANSI SQL-99 التعبير CREATE FUNCTION الذي يساعد على إنشاء التوابع.

تسمى التوابع التي يتم إنشاؤها بالتعبير CREATE FUNCTION، بالتوابع المُعرفة من قبل المستخدم أو User Defined Functions.

استدعاء تابع:

يتم استدعاء التوابع المُعرفة من قبل المستخدم بنفس الطريقة المستخدمة مع التوابع الأخرى أي يمكننا بعد إنشاء التابع استدعاؤه بصورة

التوابع في Access:

لا تدعم Access التوابع المعرفة من قبل المستخدم.

التوابع في SQL Server:

يمكن للتوابع في SQL أن تحتوي أكثر من تعبير ولكن يجب ألا تسبب أي تغيير على البيانات في قاعدة البيانات.

إنشاء التوابع

مثال:

إذا أردنا إنشاء تابع باسم formatName() الذي يقوم بإعادة الاسم الكامل مرتباً، بوضع الاسم الثاني أولاً والاسم الأول ثانياً مفصولاً بفاصلة نكتب الصيغة:

```
CREATE FUNCTION formatName (@fullName varchar(50))
RETURNS varchar(50)
AS
BEGIN
RETURN WRITE (@fullName, LEN (@fullName) – CHARINDEX (‘ ‘, @fullName)
+1) + ‘, ‘ +
LEFT (@fullName, CHARINDEX (‘ ‘, @fullName) – 1)
END;
```

نلاحظ في البداية أننا قمنا:

- بتعريف معاملات الدخل لهذا التابع، ففي حالتنا هناك معامل واحد باسم fullName،
- باستخدام التعبير RETURNS
- بتحديد أن هذا التابع سيعيد قيمة من النوع المحدد بعد RETURNS
- باستخدام التعبير RETURN لإعادة قيمة من النوع المحدد بعد RETURNS.

الآن لاستعمال هذا التابع يمكننا كتابة الصيغة:

```
SELECT formatName (‘Majd Amer’);
```

إنشاء التوابع

التوابع في Oracle:

بشكل عام، تكون الصيغة الخاصة بتعريف تابع في Oracle مشابهة لتلك المستخدمة في الإجرائيات المُخزنة في Oracle وهي من الشكل:

```
CREATE [OR REPLACE] FUNCTION function_name
(parameter_list)
RETURN data_type
IS
Variable_list
BEGIN
-- SQL Statements
RETURN expression;
END;
```

مثال:

إذا أردنا كتابة التابع الذي يقوم بأداء نفس العمل الذي قمنا بتحديده في المثال الخاص بـ SQL server فستكون الصيغة من الشكل:

```
CREATE OR REPLACE FUNCTION FormatName(FullName IN varchar)
RETURN VARCHAR
IS
FormattedName varchar(50)
BEGIN
formattedName:=
SUBSTR(FullName,INSTR(FullName,' ')+1) || ', ' ||
SUBSTR(FullName,1,INSTR(FullName,' ')-1);
RETURN(formattedName)
END;
```

نلاحظ أن نقاط الاختلاف الرئيسية تكمن هنا في أننا يجب أن نحدد نوع المعامل (IN أو OUT)، إضافةً إلى ضرورة التصريح عن المتحولات المستخدمة في الكتلة BEGIN…END كما هو الحال مع formattedName.

نلاحظ هنا عدم استخدام الإشارة @ قبل أسماء المتحولات واستخدام الإشارة:= لتعيين قيمة لمتحول.

إنشاء التوابع

التوابع في Oracle:

بشكل عام الصيغة الخاصة بتعريف تابع في Oracle مشابهة لتلك المستخدمة في الإجرائيات المخزنة في Oracle.

إنشاء التوابع

التوابع في DB2:

تشبه الصيغة الأساسية لإنشاء تابع، معرّف من قبل المستخدم في DB2، الصيغة المُعرفة في المعيار ANSI وهي من الشكل:

```
CREATE FUNCTION function_name
(parameter_list)
RETURN data_type
[LANGUAGE SQL)
[DETERMINISTIC | NON DETERMINISTIC]
[CONTAINS SQL | READS SQL DATA]
[BEGIN ATOMIC]
[SQL Statements]
RETURN expression;
[END]
```

هناك عدد لا بأس به من الخيارات التي يمكننا تحديدها:

1- يمكننا تحديد اللغة التي يكتب فيها هذا التابع. ففي حالتنا نستخدم SQL علماً أنه يمكن تخصيص لغة أخرى مثل C++ أو JAVA.

2- يمكننا أن نحدد فيما إذا كان هذا التابع سيعيد دوماً نفس القيمة مع نفس معاملات الدخل، حيث ندعوه في هذه الحالة تابع DETERMINISTIC أو تتغير القيمة المعادة رغم ثبات معاملات الدخل، حيث ندعوه في هذه الحالة NON DETERMINISTIC.

3- يمكننا تحديد فيما إذا كان التابع يقرأ بيانات من قاعدة البيانات أو أنه يستخدم عبارات SQL للتحكم بالبيانات الممررة إليه عن طريق معاملات الدخل. يتحدد هذا الأمر بالخيار READS SQL DATA أو CONTAINS SQL.

يمكن لجسم التابع أن يتكون كحد أدنى من تعبير RETURN مع القيمة المراد إعادتها أو من مجموعة من تعابير SQL بشرط إحاطتها بمحددات BEGIN ATOMIC…END.

مثال:

إذا أردنا العودة من جديد إلى المثال الذي اخترناه لتنسيق اسم شخص مدخل تصبح الصيغة في DB2 من الشكل:

```
CREATE FUNCTION formatName (fullName varchar(50))
RETURNS varchar(50)
LANGUAGE SQL
DETERMINISTIC
CONTAINS SQL
BEGIN ATOMIC
DECALRE formattedName varchar(50)
SET formattedName =
SUBSTR (fullName, POSSTR (fullName, ‘ ’) +1) || ‘, ’ ||
SUBSTR (fullName, 1, POSSTR (fullName, ‘ ’) -1);
RETURN formattedName;
END;
```

نلاحظ في المثال السابق أننا استخدمنا الخيارين DETERMINISTIC وCONTAINS SQL لكوننا نعلم أن تابعنا سيعيد القيمة نفسها من أجل نفس المدخلات وأنه لن يقوم بعمليات قراءة من قاعدة البيانات.

إنشاء التوابع

التوابع في DB2:

تشبه الصيغة الأساسية لإنشاء تابع، معرّف من قبل المستخدم في DB2، الصيغة المُعرفة في المعيار ANSI

هناك عدد لا بأس به من الخيارات التي يمكننا تحديدها:

4- يمكننا تحديد اللغة التي يكتب فيها هذا التابع. ففي حالتنا نستخدم SQL علماً أنه يمكن تخصيص لغة أخرى مثل C++ أو JAVA.

5- يمكننا أن نحدد فيما إذا كان هذا التابع سيعيد دوماً نفس القيمة مع نفس معاملات الدخل، حيث ندعوه في هذه الحالة تابع DETERMINISTIC أو تتغير القيمة المعادة رغم ثبات معاملات الدخل، حيث ندعوه في هذه الحالة NON DETERMINISTIC.

6- يمكننا تحديد فيما إذا كان التابع يقرأ بيانات من قاعدة البيانات أو أنه يستخدم عبارات SQL للتحكم بالبيانات الممررة إليه عن طريق معاملات الدخل. يتحدد هذا الأمر بالخيار READS SQL DATA أو CONTAINS SQL.

يمكن لجسم التابع أن يتكون كحد أدنى من تعبير RETURN مع القيمة المراد إعادتها أو من مجموعة من تعابير SQL بشرط إحاطتها بمحددات BEGIN ATOMIC…END.

إنشاء التوابع

التوابع في MySQL:

للأسف لا تدعم MySQL التوابع المكتوبة بلغة SQL لذلك سنحتاج عند الرغبة بإنشاء تابع معرف من قبل مستخدم، إلى كتابة نص التابع باستخدام لغة مثل C++ أو C وتسجيله في MySQL باستخدام الأمر CREATE FUNCTION الذي يأخذ الصيغة:

```
CREATE [AGGREGATE] FUNCTION function_name
RETURNS {STRING | REAL |INTEGER} SONAME chared_library_name
```

يتم استخدام التعبير AGGREGATE في حال كان التابع من النوع التجميعي مثل COUNT.

تحدد SONAME المكتبات المشتركة التي تم فيها تعريف التابع، حيث يمكن أن تكون من النمط SO أو DLL. وبعد كتابة النص البرمجي لتابعنا بأحد اللغات، نقوم بترجمته وتسجيله بالصيغة المحددة أعلاه مثلاً:

```
CREATE FUNCTION formatName
RETURNS STRING
SONAME 'C:\\MYSQL\\LIB\\MYSQLFUNCTION.DLL';
```

في حالة نظام تشغيل LINUX يجب تحديد موقع الملفات ذات اللاحقة SO بدلاً عن DLL.

التوابع في MySQL:

للأسف لا تدعم MySQL التوابع المكتوبة بلة SQL لذلك سنحتاج عند الرغبة بإنشاء تابع معرف من قبل مستخدم، إلى كتابة نص التابع باستخدام لغة مثل ++C أو C وتسجيله في MySQL باستخدام الأمر CREATE FUNCTION.

يتم استخدام التعبير AGGREGATE في حال كان التابع من النوع التجميعي مثل COUNT.

تحدد SONAME المكتبات المشتركة التي تم فيها تعريف التابع، حيث يمكن أن تكون من النمط SO أو DLL. وبعد كتابة النص البرمجي لتابعنا بأحد اللغات، نقوم بترجمته وتسجيله بالصيغة المحددة أعلاه

في حالة نظام تشغيل LINUX يجب تحديد موقع الملفات ذات اللاحقة SO بدلاً عن DLL.

قراءات اضافية

http://www-db.stanford.edu/~ullman/fcdb/oracle/my-nonstandard.html

http://www-db.stanford.edu/~ullman/fcdb/oracle/or-nonstandard.html

http://www-db.stanford.edu/~ullman/fcdb/oracle/or-plsql.html